

[illegible]

## Title: Radiologist Workstation

## Source Code Listing (Volume 2)

```

/*
 * @author Imtiaz Hossain
 *
 * @version 2.0
 *
 * Date : 10/05/00
 *
 * Header file for JPEGlib
 */

#include<iostream.h>
#include <afxwin.h>

#ifdef __cplusplus
extern "C" {
#endif // __cplusplus
#include <time.h>
#include "jpeglib.h"
#ifdef __cplusplus
}
#endif // __cplusplus

class CodecErrorHandler
{
public:
    int n_ErrorFlag;
    // Methods
    int GetFlag(){
        return n_ErrorFlag;
    }
    void SetFlag(int flag){
        n_ErrorFlag=flag;
    }
    void Notify_stderr(int flag){
        cout <<"Error: #" <<flag<<endl;
    }
};

class JPEG: public CodecErrorHandler
{
public:
    // static consts
    // Error definitions
    static const int SUCCESS;
    static const int MEMORY_ALLOC_ERROR;
    static const int FILE_READ_ERROR;
    static const int FILE_WRITE_ERROR;
    static const int JPEGLIB_STRUCT_INIT_ERROR;

    // Coding types

    static const int DEFAULT_CODING;
    static const int BASELINE;
    static const int PROGRESSIVE;
    static const int LOSSLESS;

    // Resolution
    static const int DEFAULT_RES;
    static const int ONE_BYTE;
    static const int TWO_BYTE;
    static const int THREE_BYTE;
    static const int FOUR_BYTE;

```

```

// Color Planes
static const int DEFAULT_BPP;
static const int Gray;
static const int RGB;

// Image Quality
static const int DEF_QUALITY;

// Compression Ratio
static const float DEF_RATIO;

// Time limit on the compression
static const long DEF_TIME;

// regular public variables.

int n_ImageHeight;
int n_ImageWidth;
int n_ImageBpp;
int n_ImageResolution;
int n_ImageCodingType;
int n_ImageQuality;

// Constructor (Default)
JPEG(){}

// Encoder Methods
/* The EncoderParam method loads the values for
    1) The Height of the image.
    2) The Width of the image.
    3) The number of color components per pixel, i.e. color depth.
    4) [Optional] Image resolution. Number of bits per pixel.
       This is usually a Grayscale thing. But the comprehension of the input data buffer
       (in parameter 1) will change. We usually have two options 8 or 12. DICOM will
       support upto 16.
    5) [Optional] The type of Encoding. ... i.e. BASELINE, PROGRESSIVE, etc. etc. Again
       this might call for a change in the way the decoder produces the output stream of
       BYTES.
    6) [Optional] Quality of the compression. On a scale of [0 - 100] this parameter
       determines the tradeoff between compression and image quality. For larger
       values of this parameter, we might actually have a blow-up of the size instead
       of compression. A 100 on this parameter does NOT imply lossless compression.
*/
void EncoderParam(int n_Height, int n_Width, int n_Bpp, int n_Quality=DEF_QUALITY, int n_Coding=
DEFAULT_CODING, int n_Res=DEFAULT_RES);

/* The Encode method works on the values fed into the EncoderParam method. These values
eventually show up in the modified "IJG" structure for jpeg compression. The compression
proceeds as per these values. The compressed buffer is returned by the function. The only
input parameter is a pointer to an "int", so as to avoid having to declare a "struct". The
total length of the returned buffer is written into this variable.
*/
BYTE * Encode(BYTE *jpeg_get_Buffer, int *length, int *ret_quality, int n_Height, int n_Width, i
nt n_Bpp, int n_Quality=DEF_QUALITY, int n_Res=DEFAULT_RES, int n_Coding=DEFAULT_CODING);

BYTE * Encode(BYTE *jpeg_get_Buffer, int *length, int *ret_quality, int n_Height, int n_Width,
int n_Bpp, float n_Ratio=DEF_RATIO, long n_Time=DEF_TIME, int n_Res=DEFAULT_RES, int n_Coding=DEFAUL
T_CODING);

/* The decoder needs to have certain information about its input before it processes it. The
input to the decoder is in the form of a stream of BYTES consisting of the compressed data.
The height, width and number of colors of the uncompressed image need to be specified. The
pixel-resolution also needs to be specified. The DecodeParam method just serves to collect
these values before the decoding starts.
*/

```





```

/*
 * jpegint.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file provides common declarations for the various JPEG modules.
 * These declarations are considered internal to the JPEG library; most
 * applications using the library shouldn't need to include this file.
 */

/* Declarations for both compression & decompression */

typedef enum (
    JBUF_PASS_THRU, /* Operating modes for buffer controllers */
    /* Plain stripwise operation */
    /* Remaining modes require a full-image buffer to have been created */
    JBUF_SAVE_SOURCE, /* Run source subobject only, save output */
    JBUF_CRANK_DEST, /* Run dest subobject only, using saved data */
    JBUF_SAVE_AND_PASS /* Run both subobjects, save output */
) J_BUF_MODE;

/* Values of global_state field (jdapi.c has some dependencies on ordering!) */
#define CSTATE_START 100 /* after create_compress */
#define CSTATE_SCANNING 101 /* start_compress done, write_scanlines OK */
#define CSTATE_RAW_OK 102 /* start_compress done, write_raw_data OK */
#define CSTATE_WRCOEFS 103 /* jpeg_write_coefficients done */
#define DSTATE_START 200 /* after create_decompress */
#define DSTATE_INHEADER 201 /* reading header markers, no SOS yet */
#define DSTATE_READY 202 /* found SOS, ready for start_decompress */
#define DSTATE_PRELOAD 203 /* reading multiscan file in start_decompress */
#define DSTATE_PRESCAN 204 /* performing dummy pass for 2-pass quant */
#define DSTATE_SCANNING 205 /* start_decompress done, read_scanlines OK */
#define DSTATE_RAW_OK 206 /* start_decompress done, read_raw_data OK */
#define DSTATE_BUFIMAGE 207 /* expecting jpeg_start_output */
#define DSTATE_BUFPOST 208 /* looking for SOS/EOI in jpeg_finish_output */
#define DSTATE_RDCOEFS 209 /* reading file in jpeg_read_coefficients */
#define DSTATE_STOPPING 210 /* looking for EOI in jpeg_finish_decompress */

/* Declarations for compression modules */

/* Master control module */
struct jpeg_comp_master {
    JMETHOD(void, prepare_for_pass, (j_compress_ptr cinfo));
    JMETHOD(void, pass_startup, (j_compress_ptr cinfo));
    JMETHOD(void, finish_pass, (j_compress_ptr cinfo));

    /* State variables made visible to other modules */
    boolean call_pass_startup; /* True if pass_startup must be called */
    boolean is_last_pass; /* True during last pass */
};

/* Main buffer control (downsampled-data buffer) */
struct jpeg_c_main_controller {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, process_data, (j_compress_ptr cinfo,
        JSAMPARRAY input_buf, JDIMENSION *in_row_ctr,
        JDIMENSION in_rows_avail));
};

/* Compression preprocessing (downsampling input buffer control) */
struct jpeg_c_prep_controller {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, pre_process_data, (j_compress_ptr cinfo,
        JSAMPARRAY input_buf,
        JDIMENSION *in_row_ctr,
        JDIMENSION in_rows_avail,
        JSAMPIMAGE output_buf,
        JDIMENSION *out_row_group_ctr,
        JDIMENSION out_row_groups_avail));
};

/* Coefficient buffer control */
struct jpeg_c_coef_controller {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, compress_data, (j_compress_ptr cinfo,
        JSAMPIMAGE input_buf));
};

```

```

/* Colorspace conversion */
struct jpeg_color_converter {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo));
    JMETHOD(void, color_convert, (j_compress_ptr cinfo,
        JSAMPARRAY input_buf, JSAMPIMAGE output_buf,
        JDIMENSION output_row, int num_rows));
};

/* Downsampling */
struct jpeg_downsampler {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo));
    JMETHOD(void, downsample, (j_compress_ptr cinfo,
        JSAMPIMAGE input_buf, JDIMENSION in_row_index,
        JSAMPIMAGE output_buf,
        JDIMENSION out_row_group_index));

    boolean need_context_rows; /* TRUE if need rows above & below */
};

/* Forward DCT (also controls coefficient quantization) */
struct jpeg_forward_dct {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo));
    /* perhaps this should be an array??? */
    JMETHOD(void, forward_DCT, (j_compress_ptr cinfo,
        jpeg_component_info * comp_ptr,
        JSAMPARRAY sample_data, JBLOCKROW coef_blocks,
        JDIMENSION start_row, JDIMENSION start_col,
        JDIMENSION num_blocks));
};

/* Entropy encoding */
struct jpeg_entropy_encoder {
    JMETHOD(void, start_pass, (j_compress_ptr cinfo, boolean gather_statistics));
    JMETHOD(boolean, encode_mcu, (j_compress_ptr cinfo, JBLOCKROW *MCU_data));
    JMETHOD(void, finish_pass, (j_compress_ptr cinfo));
};

/* Marker writing */
struct jpeg_marker_writer {
    JMETHOD(void, write_file_header, (j_compress_ptr cinfo));
    JMETHOD(void, write_frame_header, (j_compress_ptr cinfo));
    JMETHOD(void, write_scan_header, (j_compress_ptr cinfo));
    JMETHOD(void, write_file_trailer, (j_compress_ptr cinfo));
    JMETHOD(void, write_tables_only, (j_compress_ptr cinfo));
    /* These routines are exported to allow insertion of extra markers */
    /* Probably only COM and APPn markers should be written this way */
    JMETHOD(void, write_marker_header, (j_compress_ptr cinfo, int marker,
        unsigned int datalen));
    JMETHOD(void, write_marker_byte, (j_compress_ptr cinfo, int val));
};

/* Declarations for decompression modules */

/* Master control module */
struct jpeg_decomp_master {
    JMETHOD(void, prepare_for_output_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, finish_output_pass, (j_decompress_ptr cinfo));

    /* State variables made visible to other modules */
    boolean is_dummy_pass; /* True during 1st pass for 2-pass quant */
};

/* Input control module */
struct jpeg_input_controller {
    JMETHOD(int, consume_input, (j_decompress_ptr cinfo));
    JMETHOD(void, reset_input_controller, (j_decompress_ptr cinfo));
    JMETHOD(void, start_input_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, finish_input_pass, (j_decompress_ptr cinfo));

    /* State variables made visible to other modules */
    boolean has_multiple_scans; /* True if file has multiple scans */
    boolean eoi_reached; /* True when EOI has been consumed */
};

/* Main buffer control (downsampled-data buffer) */
struct jpeg_d_main_controller {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, process_data, (j_decompress_ptr cinfo,

```

```

        JSAMPARRAY output_buf, JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail));
};

/* Coefficient buffer control */
struct jpeg_d_coef_controller {
    JMETHOD(void, start_input_pass, (j_decompress_ptr cinfo));
    JMETHOD(int, consume_data, (j_decompress_ptr cinfo));
    JMETHOD(void, start_output_pass, (j_decompress_ptr cinfo));
    JMETHOD(int, decompress_data, (j_decompress_ptr cinfo,
        JSAMPIMAGE output_buf));
    /* Pointer to array of coefficient virtual arrays, or NULL if none */
    jvirt_barray_ptr *coef_arrays;
};

/* Decompression postprocessing (color quantization buffer control) */
struct jpeg_d_post_controller {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo, J_BUF_MODE pass_mode));
    JMETHOD(void, post_process_data, (j_decompress_ptr cinfo,
        JSAMPIMAGE input_buf,
        JDIMENSION *in_row_group_ctr,
        JDIMENSION in_row_groups_avail,
        JSAMPARRAY output_buf,
        JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail));
};

/* Marker reading & parsing */
struct jpeg_marker_reader {
    JMETHOD(void, reset_marker_reader, (j_decompress_ptr cinfo));
    /* Read markers until SOS or EOI.
     * Returns same codes as are defined for jpeg_consume_input:
     * JPEG_SUSPENDED, JPEG_REACHED_SOS, or JPEG_REACHED_EOI.
     */
    JMETHOD(int, read_markers, (j_decompress_ptr cinfo));
    /* Read a restart marker --- exported for use by entropy decoder only */
    jpeg_marker_parser_method read_restart_marker;

    /* State of marker reader --- nominally internal, but applications
     * supplying COM or APPn handlers might like to know the state.
     */
    boolean saw_SOI; /* found SOI? */
    boolean saw_SOF; /* found SOF? */
    int next_restart_num; /* next restart number expected (0-7) */
    unsigned int discarded_bytes; /* # of bytes skipped looking for a marker */
};

/* Entropy decoding */
struct jpeg_entropy_decoder {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    JMETHOD(boolean, decode_mcu, (j_decompress_ptr cinfo,
        JBLOCKROW *MCU_data));

    /* This is here to share code between baseline and progressive decoders; */
    /* other modules probably should not use it */
    boolean insufficient_data; /* set TRUE after emitting warning */
};

/* Inverse DCT (also performs dequantization) */
typedef JMETHOD(void, inverse_DCT_method_ptr,
    (j_decompress_ptr cinfo, jpeg_component_info * comp_ptr,
    JCOEFPTR coef_block,
    JSAMPARRAY output_buf, JDIMENSION output_col));

struct jpeg_inverse_dct {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    /* It is useful to allow each component to have a separate IDCT method. */
    inverse_DCT_method_ptr inverse_DCT[MAX_COMPONENTS];
};

/* Upsampling (note that upsampler must also call color converter) */
struct jpeg_upsampler {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, upsample, (j_decompress_ptr cinfo,
        JSAMPIMAGE input_buf,
        JDIMENSION *in_row_group_ctr,
        JDIMENSION in_row_groups_avail,
        JSAMPARRAY output_buf,
        JDIMENSION *out_row_ctr,
        JDIMENSION out_rows_avail));
};

```

```

boolean need_context_rows; TRUE if need rows above & below
};

/* Colorspace conversion */
struct jpeg_color_deconverter {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, color_convert, (j_decompress_ptr cinfo,
        JSAMPIMAGE input_buf, JDIMENSION input_row,
        JSAMPARRAY output_buf, int num_rows));
};

/* Color quantization or color precision reduction */
struct jpeg_color_quantizer {
    JMETHOD(void, start_pass, (j_decompress_ptr cinfo, boolean is_pre_scan));
    JMETHOD(void, color_quantize, (j_decompress_ptr cinfo,
        JSAMPARRAY input_buf, JSAMPARRAY output_buf,
        int num_rows));
    JMETHOD(void, finish_pass, (j_decompress_ptr cinfo));
    JMETHOD(void, new_color_map, (j_decompress_ptr cinfo));
};

/* Miscellaneous useful macros */

#undef MAX
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#undef MIN
#define MIN(a,b) ((a) < (b) ? (a) : (b))

/* We assume that right shift corresponds to signed division by 2 with
rounding towards minus infinity. This is correct for typical "arithmetic
shift" instructions that shift in copies of the sign bit. But some
C compilers implement >> with an unsigned shift. For these machines you
must define RIGHT_SHIFT_IS_UNSIGNED.
RIGHT_SHIFT provides a proper signed right shift of an INT32 quantity.
It is only applied with constant shift counts. SHIFT_TEMPS must be
included in the variables of any routine using RIGHT_SHIFT.
*/
#ifdef RIGHT_SHIFT_IS_UNSIGNED
#define SHIFT_TEMPS INT32 shift_temp;
#define RIGHT_SHIFT(x,shft) \
    ((shift_temp = (x)) < 0 ? \
    (shift_temp >> (shft)) | ((~((INT32) 0)) << (32-(shft))) : \
    (shift_temp >> (shft)))
#else
#define SHIFT_TEMPS
#define RIGHT_SHIFT(x,shft) ((x) >> (shft))
#endif

/* Short forms of external names for systems with brain-damaged linkers. */

#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jinit_compress_master jICompress
#define jinit_c_master_control jICMaster
#define jinit_c_main_controller jICMainC
#define jinit_c_prep_controller jICPrepC
#define jinit_c_coef_controller jICCoefC
#define jinit_color_converter jICColor
#define jinit_downsampler jIDownsampler
#define jinit_forward_dct jIFDCT
#define jinit_huff_encoder jIHEncoder
#define jinit_phuff_encoder jIPHEncoder
#define jinit_marker_writer jIMWriter
#define jinit_master_decompress jIDMaster
#define jinit_d_main_controller jIDMainC
#define jinit_d_coef_controller jIDCoefC
#define jinit_d_post_controller jIDPostC
#define jinit_input_controller jIInCtlr
#define jinit_marker_reader jIMReader
#define jinit_huff_decoder jIHDecoder
#define jinit_phuff_decoder jIPHDecoder
#define jinit_inverse_dct jIIDCT
#define jinit_upsampler jIUpsampler
#define jinit_color_deconverter jIDColor
#define jinit_lpass_quantizer jI1Quant
#define jinit_2pass_quantizer jI2Quant

```

```

#define jinit_merged_upsampler JMUpsampler
#define jinit_memory_mgr jmmgr
#define jdiv_round_up jDivRound
#define jround_up jRound
#define jcopy_sample_rows jCopySamples
#define jcopy_block_row jCopyBlocks
#define jzero_far jZeroFar
#define jpeg_zigzag_order jZIGTable
#define jpeg_natural_order jZAGTable
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Compression module initialization routines */
EXTERN(void) jinit_compress_master JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_c_master_control JPP((j_compress_ptr cinfo,
    boolean transcode_only));
EXTERN(void) jinit_c_main_controller JPP((j_compress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_c_prep_controller JPP((j_compress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_c_coef_controller JPP((j_compress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_color_converter JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_downsampler JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_forward_dct JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_huff_encoder JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_phuff_encoder JPP((j_compress_ptr cinfo));
EXTERN(void) jinit_marker_writer JPP((j_compress_ptr cinfo));
/* Decompression module initialization routines */
EXTERN(void) jinit_master_decompress JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_d_main_controller JPP((j_decompress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_d_coef_controller JPP((j_decompress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_d_post_controller JPP((j_decompress_ptr cinfo,
    boolean need_full_buffer));
EXTERN(void) jinit_input_controller JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_marker_reader JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_huff_decoder JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_phuff_decoder JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_inverse_dct JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_upsampler JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_color_deconverter JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_lpass_quantizer JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_2pass_quantizer JPP((j_decompress_ptr cinfo));
EXTERN(void) jinit_merged_upsampler JPP((j_decompress_ptr cinfo));
/* Memory manager initialization */
EXTERN(void) jinit_memory_mgr JPP((j_common_ptr cinfo));

/* Utility routines in jutils.c */
EXTERN(long) jdiv_round_up JPP((long a, long b));
EXTERN(long) jround_up JPP((long a, long b));
EXTERN(void) jcopy_sample_rows JPP((JSAMPARRAY input_array, int source_row,
    JSAMPARRAY output_array, int dest_row,
    int num_rows, JDIMENSION num_cols));
EXTERN(void) jcopy_block_row JPP((JBLOCKROW input_row, JBLOCKROW output_row,
    JDIMENSION num_blocks));
EXTERN(void) jzero_far JPP((void * target, size_t bytestozero));
/* Constant tables in jutils.c */
#if 0
    /* This table is not actually needed in v6a */
extern const int jpeg_zigzag_order[]; /* natural coef order to zigzag order */
#endif
extern const int jpeg_natural_order[]; /* zigzag coef order to natural order */

/* Suppress undefined-structure complaints if necessary. */

#ifdef INCOMPLETE_TYPES_BROKEN
#ifdef AM_MEMORY_MANAGER /* only jmemmgr.c defines these */
struct jvirt_sarray_control { long dummy; };
struct jvirt_barray_control { long dummy; };
#endif
#endif
#endif /* INCOMPLETE_TYPES_BROKEN */

```

```

/*
 * jpeglib.h
 *
 * Copyright (C) 1991-1998, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file defines the application interface for the JPEG library.
 * Most applications using the library need only include this file,
 * and perhaps jerror.h if they want to know the exact error codes.
 */

#ifndef JPEGLIB_H
#define JPEGLIB_H

/*
 * First we include the configuration files that record how this
 * installation of the JPEG library is set up.  jconfig.h can be
 * generated automatically for many systems.  jmorecfg.h contains
 * manual configuration options that most people need not worry about.
 */

#ifndef JCONFIG_INCLUDED /* in case jinclude.h already did */
#include "jconfig.h" /* widely used configuration options */
#endif
#include "jmorecfg.h" /* seldom changed options */

#ifndef STDIO_H
#include <stdio.h>
#define STDIO_H
#endif

/*
 * Version ID for the JPEG library.
 * This might be useful for tests like "#if JPEG_LIB_VERSION >= 60".
 */
#define JPEG_LIB_VERSION 62 /* Version 6b */

/*
 * Various constants determining the sizes of things.
 * All of these are specified by the JPEG standard, so don't change them
 * if you want to be compatible.
 */

#define DCTSIZE 8 /* The basic DCT block is 8x8 samples */
#define DCTSIZE2 64 /* DCTSIZE squared; # of elements in a block */
#define NUM_QUANT_TBLS 4 /* Quantization tables are numbered 0..3 */
#define NUM_HUFF_TBLS 4 /* Huffman tables are numbered 0..3 */
#define NUM_ARITH_TBLS 16 /* Arith-coding tables are numbered 0..15 */
#define MAX_COMPS_IN_SCAN 4 /* JPEG limit on # of components in one scan */
#define MAX_SAMP_FACTOR 4 /* JPEG limit on sampling factors */

/* Unfortunately, some bozo at Adobe saw no reason to be bound by the standard;
 * the PostScript DCT filter can emit files with many more than 10 blocks/MCU.
 * If you happen to run across such a file, you can up D_MAX_BLOCKS_IN_MCU
 * to handle it. We even let you do this from the jconfig.h file. However,
 * we strongly discourage changing C_MAX_BLOCKS_IN_MCU; just because Adobe
 * sometimes emits noncompliant files doesn't mean you should too.
 */
#define C_MAX_BLOCKS_IN_MCU 10 /* compressor's limit on blocks per MCU */
#ifndef D_MAX_BLOCKS_IN_MCU
#define D_MAX_BLOCKS_IN_MCU 10 /* decompressor's limit on blocks per MCU */
#endif

/*
 * Data structures for images (arrays of samples and of DCT coefficients).
 * On 80x86 machines, the image arrays are too big for near pointers,
 * but the pointer arrays can fit in near memory.
 */

typedef JSAMPLE *JSAMPROW; /* ptr to one image row of pixel samples. */
typedef JSAMPROW *JSAMPARRAY; /* ptr to some rows (a 2-D sample array) */
typedef JSAMPARRAY *JSAMPIMAGE; /* a 3-D sample array: top index is color */

typedef JCOEF JBLOCK[DCTSIZE2]; /* one block of coefficients */
typedef JBLOCK *JBLOCKROW; /* pointer to one row of coefficient blocks */
typedef JBLOCKROW *JBLOCKARRAY; /* a 2-D array of coefficient blocks */
typedef JBLOCKARRAY *JBLOCKIMAGE; /* a 3-D array of coefficient blocks */

typedef JCOEF *JCOEFPTR; /* useful in a couple of places */

```

```

/* Types for JPEG compression parameters and working tables. */

/* DCT coefficient quantization tables. */

typedef struct {
    /* This array gives the coefficient quantizers in natural array order
     * (not the zigzag order in which they are stored in a JPEG DQT marker).
     * CAUTION: IJG versions prior to v6a kept this array in zigzag order.
     */
    UINT16 quantval[DCTSIZE2]; /* quantization step for each coefficient */
    /* This field is used only during compression. It's initialized FALSE when
     * the table is created, and set TRUE when it's been output to the file.
     * You could suppress output of a table by setting this to TRUE.
     * (See jpeg_suppress_tables for an example.)
     */
    boolean sent_table; /* TRUE when table has been output */
} JQUANT_TBL;

/* Huffman coding tables. */

typedef struct {
    /* These two fields directly represent the contents of a JPEG DHT marker */
    UINT8 bits[17]; /* bits[k] = # of symbols with codes of */
    /* length k bits; bits[0] is unused */
    UINT8 huffval[256]; /* The symbols, in order of incr code length */
    /* This field is used only during compression. It's initialized FALSE when
     * the table is created, and set TRUE when it's been output to the file.
     * You could suppress output of a table by setting this to TRUE.
     * (See jpeg_suppress_tables for an example.)
     */
    boolean sent_table; /* TRUE when table has been output */
} JHUFF_TBL;

/* Basic info about one component (color channel). */

typedef struct {
    /* These values are fixed over the whole image. */
    /* For compression, they must be supplied by parameter setup; */
    /* for decompression, they are read from the SOF marker. */
    int component_id; /* identifier for this component (0..255) */
    int component_index; /* its index in SOF or cinfo->comp_info[] */
    int h_samp_factor; /* horizontal sampling factor (1..4) */
    int v_samp_factor; /* vertical sampling factor (1..4) */
    int quant_tbl_no; /* quantization table selector (0..3) */
    /* These values may vary between scans. */
    /* For compression, they must be supplied by parameter setup; */
    /* for decompression, they are read from the SOS marker. */
    /* The decompressor output side may not use these variables. */
    int dc_tbl_no; /* DC entropy table selector (0..3) */
    int ac_tbl_no; /* AC entropy table selector (0..3) */

    /* Remaining fields should be treated as private by applications. */

    /* These values are computed during compression or decompression startup: */
    /* Component's size in DCT blocks.
     * Any dummy blocks added to complete an MCU are not counted; therefore
     * these values do not depend on whether a scan is interleaved or not.
     */
    JDIMENSION width_in_blocks;
    JDIMENSION height_in_blocks;
    /* Size of a DCT block in samples. Always DCTSIZE for compression.
     * For decompression this is the size of the output from one DCT block,
     * reflecting any scaling we choose to apply during the IDCT step.
     * Values of 1,2,4,8 are likely to be supported. Note that different
     * components may receive different IDCT scalings.
     */
    int DCT_scaled_size;
    /* The downsampled dimensions are the component's actual, unpadded number
     * of samples at the main buffer (preprocessing/compression interface), thus
     * downsampled_width = ceil(image_width * Hi/Hmax)
     * and similarly for height. For decompression, IDCT scaling is included, so
     * downsampled_width = ceil(image_width * Hi/Hmax * DCT_scaled_size/DCTSIZE)
     */
    JDIMENSION downsampled_width; /* actual width in samples */
    JDIMENSION downsampled_height; /* actual height in samples */
    /* This flag is used only for decompression. In cases where some of the

```

```

* components will be ignored. eg grayscale output from YCbCr image.
* we can skip most computations for the unused components.
*/
boolean component_needed; /* do we need the value of this component? */

/* These values are computed before starting a scan of the component. */
/* The decompressor output side may not use these variables. */
int MCU_width; /* number of blocks per MCU, horizontally */
int MCU_height; /* number of blocks per MCU, vertically */
int MCU_blocks; /* MCU_width * MCU_height */
int MCU_sample_width; /* MCU width in samples, MCU_width*DCT_scaled_size */
int last_col_width; /* # of non-dummy blocks across in last MCU */
int last_row_height; /* # of non-dummy blocks down in last MCU */

/* Saved quantization table for component; NULL if none yet saved.
* See jinput.c comments about the need for this information.
* This field is currently used only for decompression.
*/
JQUANT_TBL * quant_table;

/* Private per-component storage for DCT or IDCT subsystem. */
void * dct_table;
} jpeg_component_info;

/* The script for encoding a multiple-scan file is an array of these: */
typedef struct {
    int comps_in_scan; /* number of components encoded in this scan */
    int component_index[MAX_COMPS_IN_SCAN]; /* their SOF/comp_info[] indexes */
    int Ss, Se; /* progressive JPEG spectral selection parms */
    int Ah, Al; /* progressive JPEG successive approx. parms */
} jpeg_scan_info;

/* The decompressor can save APPn and COM markers in a list of these: */
typedef struct jpeg_marker_struct * jpeg_saved_marker_ptr;

struct jpeg_marker_struct {
    jpeg_saved_marker_ptr next; /* next in list, or NULL */
    UINT8 marker; /* marker code: JPEG_COM, or JPEG_APP0+n */
    unsigned int original_length; /* # bytes of data in the file */
    unsigned int data_length; /* # bytes of data saved at data[] */
    JOCTET * data; /* the data contained in the marker */
    /* the marker length word is not counted in data_length or original_length */
} jpeg_marker_struct;

/* Known color spaces. */
typedef enum {
    JCS_UNKNOWN, /* error/unspecified */
    JCS_GRAYSCALE, /* monochrome */
    JCS_RGB, /* red/green/blue */
    JCS_YCbCr, /* Y/Cb/Cr (also known as YUV) */
    JCS_CMYK, /* C/M/Y/K */
    JCS_YCCK, /* Y/Cb/Cr/K */
} J_COLOR_SPACE;

/* DCT/IDCT algorithm options. */
typedef enum {
    JDCT_ISLOW, /* slow but accurate integer algorithm */
    JDCT_IFAST, /* faster, less accurate integer method */
    JDCT_FLOAT, /* floating-point: accurate, fast on fast HW */
} J_DCT_METHOD;

#ifdef JDCT_DEFAULT /* may be overridden in jconfig.h */
#define JDCT_DEFAULT JDCT_ISLOW
#endif
#ifdef JDCT_FASTEST /* may be overridden in jconfig.h */
#define JDCT_FASTEST JDCT_IFAST
#endif

/* Dithering options for decompression. */
typedef enum {
    JDITHER_NONE, /* no dithering */
    JDITHER_ORDERED, /* simple ordered dither */
    JDITHER_FS, /* Floyd-Steinberg error diffusion dither */
} J_DITHER_MODE;

```



```

/* Common fields between JPEG compression and decompression master structs. */

#define jpeg_common_fields \
    struct jpeg_error_mgr * err; /* Error handler module */\
    struct jpeg_memory_mgr * mem; /* Memory manager module */\
    struct jpeg_progress_mgr * progress; /* Progress monitor, or NULL if none */\
    void * client_data; /* Available for use by application */\
    boolean is_decompressor; /* So common code can tell which is which */\
    int global_state /* For checking call sequence validity */

/* Routines that are to be used by both halves of the library are declared
 * to receive a pointer to this structure. There are no actual instances of
 * jpeg_common_struct, only of jpeg_compress_struct and jpeg_decompress_struct.
 */
struct jpeg_common_struct {
    jpeg_common_fields; /* Fields common to both master struct types */
    /* Additional fields follow in an actual jpeg_compress_struct or
     * jpeg_decompress_struct. All three structs must agree on these
     * initial fields! (This would be a lot cleaner in C++.)
     */
};

typedef struct jpeg_common_struct * j_common_ptr;
typedef struct jpeg_compress_struct * j_compress_ptr;
typedef struct jpeg_decompress_struct * j_decompress_ptr;

/* Master record for a compression instance */
struct jpeg_compress_struct {
    jpeg_common_fields; /* Fields shared with jpeg_decompress_struct */

    /* Destination for compressed data */
    struct jpeg_destination_mgr * dest;

    /* Description of source image --- these fields must be filled in by
     * outer application before starting compression. in_color_space must
     * be correct before you can even call jpeg_set_defaults().
     */
    JDIMENSION image_width; /* input image width */
    JDIMENSION image_height; /* input image height */
    int input_components; /* # of color components in input image */
    J_COLOR_SPACE in_color_space; /* colorspace of input image */

    double input_gamma; /* image gamma of input image */

    /* Compression parameters --- these fields must be set before calling
     * jpeg_start_compress(). We recommend calling jpeg_set_defaults() to
     * initialize everything to reasonable defaults, then changing anything
     * the application specifically wants to change. That way you won't get
     * burnt when new parameters are added. Also note that there are several
     * helper routines to simplify changing parameters.
     */
    /* ### I added this index variable as a counter to the compressed buffer.*/
    int index;

    int data_precision; /* bits of precision in image data */

    int num_components; /* # of color components in JPEG image */
    J_COLOR_SPACE jpeg_color_space; /* colorspace of JPEG image */

    jpeg_component_info * comp_info;
    /* comp_info[i] describes component that appears i'th in SOF */

    JQUANT_TBL * quant_tbl_ptrs[NUM_QUANT_TBLS];
    /* ptrs to coefficient quantization tables, or NULL if not defined */

    JHUFF_TBL * dc_huff_tbl_ptrs[NUM_HUFF_TBLS];
    JHUFF_TBL * ac_huff_tbl_ptrs[NUM_HUFF_TBLS];
    /* ptrs to Huffman coding tables, or NULL if not defined */

    UINT8 arith_dc_L[NUM_ARITH_TBLS]; /* L values for DC arith-coding tables */
    UINT8 arith_dc_U[NUM_ARITH_TBLS]; /* U values for DC arith-coding tables */
    UINT8 arith_ac_K[NUM_ARITH_TBLS]; /* Kx values for AC arith-coding tables */

    int num_scans; /* # of entries in scan_info array */

```

```

const jpeg_scan_info * scan_info; /* script for multi-scan file, NULL */
/* The default value of scan_info is NULL, which causes a single
 * sequential JPEG file to be emitted. To create a multi-scan file,
 * set num_scans and scan_info to point to an array of scan definitions.
 */

boolean raw_data_in;      /* TRUE=caller supplies downsampled data */
boolean arith_code;       /* TRUE=arithmetic coding, FALSE=Huffman */
boolean optimize_coding; /* TRUE=optimize entropy encoding parms */
boolean CCIR601_sampling; /* TRUE=first samples are cosited */
int smoothing_factor;     /* 1..100, or 0 for no input smoothing */
J_DCT_METHOD dct_method; /* DCT algorithm selector */

/* The restart interval can be specified in absolute MCUs by setting
 * restart_interval, or in MCU rows by setting restart_in_rows
 * (in which case the correct restart_interval will be figured
 * for each scan).
 */
unsigned int restart_interval; /* MCUs per restart, or 0 for no restart */
int restart_in_rows;          /* if > 0, MCU rows per restart interval */

/* Parameters controlling emission of special markers. */

boolean write_JFIF_header; /* should a JFIF marker be written? */
UINT8 JFIF_major_version; /* What to write for the JFIF version number */
UINT8 JFIF_minor_version;
/* These three values are not used by the JPEG code, merely copied */
/* into the JFIF APP0 marker. density_unit can be 0 for unknown, */
/* 1 for dots/inch, or 2 for dots/cm. Note that the pixel aspect */
/* ratio is defined by X_density/Y_density even when density_unit=0. */
UINT8 density_unit; /* JFIF code for pixel size units */
UINT16 X_density; /* Horizontal pixel density */
UINT16 Y_density; /* Vertical pixel density */
boolean write_Adobe_marker; /* should an Adobe marker be written? */

/* State variable: index of next scanline to be written to
 * jpeg_write_scanlines(). Application may use this to control its
 * processing loop, e.g., "while (next_scanline < image_height)".
 */
JDIMENSION next_scanline; /* 0 .. image_height-1 */

/* Remaining fields are known throughout compressor, but generally
 * should not be touched by a surrounding application.
 */

/* These fields are computed during compression startup
 */
boolean progressive_mode; /* TRUE if scan script uses progressive mode */
int max_h_samp_factor; /* largest h_samp_factor */
int max_v_samp_factor; /* largest v_samp_factor */

JDIMENSION total_iMCU_rows; /* # of iMCU rows to be input to coef ctlr */
/* The coefficient controller receives data in units of MCU rows as defined
 * for fully interleaved scans (whether the JPEG file is interleaved or not).
 * There are v_samp_factor * DCTSIZE sample rows of each component in an
 * "iMCU" (interleaved MCU) row.
 */

/*
 * These fields are valid during any one scan.
 * They describe the components and MCUs actually appearing in the scan.
 */
int comps_in_scan; /* # of JPEG components in this scan */
jpeg_component_info * cur_comp_info[MAX_COMPS_IN_SCAN];
/* *cur_comp_info[i] describes component that appears i'th in SOS */

JDIMENSION MCUs_per_row; /* # of MCUs across the image */
JDIMENSION MCU_rows_in_scan; /* # of MCU rows in the image */

int blocks_in_MCU; /* # of DCT blocks per MCU */
int MCU_membership[C_MAX_BLOCKS_IN_MCU];
/* MCU_membership[i] is index in cur_comp_info of component owning */
/* i'th block in an MCU */

int Ss, Se, Ah, Al; /* progressive JPEG parameters for scan */

/*
 * Links to compression subobjects (methods and private variables of modules)
 */

```

```

*/
struct jpeg_comp_master * master;
struct jpeg_c_main_controller * main;
struct jpeg_c_prep_controller * prep;
struct jpeg_c_coef_controller * coef;
struct jpeg_marker_writer * marker;
struct jpeg_color_converter * cconvert;
struct jpeg_downsampler * downsampler;
struct jpeg_forward_dct * fdct;
struct jpeg_entropy_encoder * entropy;
jpeg_scan_info * script_space; /* workspace for jpeg_simple_progression */
int script_space_size;
};

/* Master record for a decompression instance */

struct jpeg_decompress_struct {
    jpeg_common_fields; /* Fields shared with jpeg_compress_struct */

    /* Source of compressed data */
    struct jpeg_source_mgr * src;

    /* Basic description of image --- filled in by jpeg_read_header(). */
    /* Application may inspect these values to decide how to process image. */

    JDIMENSION image_width; /* nominal image width (from SOF marker) */
    JDIMENSION image_height; /* nominal image height */
    int num_components; /* # of color components in JPEG image */
    J_COLOR_SPACE jpeg_color_space; /* colorspace of JPEG image */

    /* Decompression processing parameters --- these fields must be set before
    * calling jpeg_start_decompress(). Note that jpeg_read_header() initializes
    * them to default values.
    */

    J_COLOR_SPACE out_color_space; /* colorspace for output */

    unsigned int scale_num, scale_denom; /* fraction by which to scale image */

    double output_gamma; /* image gamma wanted in output */

    boolean buffered_image; /* TRUE=multiple output passes */
    boolean raw_data_out; /* TRUE=downsampled data wanted */

    J_DCT_METHOD dct_method; /* IDCT algorithm selector */
    boolean do_fancy_upsampling; /* TRUE=apply fancy upsampling */
    boolean do_block_smoothing; /* TRUE=apply interblock smoothing */

    boolean quantize_colors; /* TRUE=colormapped output wanted */
    /* the following are ignored if not quantize_colors: */
    J_DITHER_MODE dither_mode; /* type of color dithering to use */
    boolean two_pass_quantize; /* TRUE=use two-pass color quantization */
    int desired_number_of_colors; /* max # colors to use in created colormap */
    /* these are significant only in buffered-image mode: */
    boolean enable_1pass_quant; /* enable future use of 1-pass quantizer */
    boolean enable_external_quant; /* enable future use of external colormap */
    boolean enable_2pass_quant; /* enable future use of 2-pass quantizer */

    /* Description of actual output image that will be returned to application.
    * These fields are computed by jpeg_start_decompress().
    * You can also use jpeg_calc_output_dimensions() to determine these values
    * in advance of calling jpeg_start_decompress().
    */

    JDIMENSION output_width; /* scaled image width */
    JDIMENSION output_height; /* scaled image height */
    int out_color_components; /* # of color components in out_color_space */
    int output_components; /* # of color components returned */
    /* output_components is 1 (a colormap index) when quantizing colors;
    * otherwise it equals out_color_components.
    */
    int rec_outbuf_height; /* min recommended height of scanline buffer */
    /* If the buffer passed to jpeg_read_scanlines() is less than this many rows
    * high, space and time will be wasted due to unnecessary data copying.
    * Usually rec_outbuf_height will be 1 or 2, at most 4.
    */

    /* When quantizing colors, the output colormap is described by these fields.
    * The application can supply a colormap by setting colormap non-NULL before

```

```

* calling jpeg_start_decompress; otherwise a colormap is created during
* jpeg_start_decompress or jpeg_start_output.
* The map has out_color_components rows and actual_number_of_colors columns.
*/
int actual_number_of_colors; /* number of entries in use */
JSAMPARRAY colormap; /* The color map as a 2-D pixel array */

/* State variables: these variables indicate the progress of decompression.
* The application may examine these but must not modify them.
*/

/* Row index of next scanline to be read from jpeg_read_scanlines().
* Application may use this to control its processing loop, e.g.,
* "while (output_scanline < output_height)".
*/
JDIMENSION output_scanline; /* 0 .. output_height-1 */

/* Current input scan number and number of iMCU rows completed in scan.
* These indicate the progress of the decompressor input side.
*/
int input_scan_number; /* Number of SOS markers seen so far */
JDIMENSION input_iMCU_row; /* Number of iMCU rows completed */

/* The "output scan number" is the notional scan being displayed by the
* output side. The decompressor will not allow output scan/row number
* to get ahead of input scan/row, but it can fall arbitrarily far behind.
*/
int output_scan_number; /* Nominal scan number being displayed */
JDIMENSION output_iMCU_row; /* Number of iMCU rows read */

/* Current progression status. coef_bits[c][i] indicates the precision
* with which component c's DCT coefficient i (in zigzag order) is known.
* It is -1 when no data has yet been received, otherwise it is the point
* transform (shift) value for the most recent scan of the coefficient
* (thus, 0 at completion of the progression).
* This pointer is NULL when reading a non-progressive file.
*/
int (*coef_bits)[DCTSIZE2]; /* -1 or current Al value for each coef */

/* Internal JPEG parameters --- the application usually need not look at
* these fields. Note that the decompressor output side may not use
* any parameters that can change between scans.
*/

/* Quantization and Huffman tables are carried forward across input
* datastreams when processing abbreviated JPEG datastreams.
*/

JQUANT_TBL * quant_tbl_ptrs[NUM_QUANT_TBLS];
/* ptrs to coefficient quantization tables, or NULL if not defined */

JHUFF_TBL * dc_huff_tbl_ptrs[NUM_HUFF_TBLS];
JHUFF_TBL * ac_huff_tbl_ptrs[NUM_HUFF_TBLS];
/* ptrs to Huffman coding tables, or NULL if not defined */

/* These parameters are never carried across datastreams, since they
* are given in SOF/SOS markers or defined to be reset by SOI.
*/

int data_precision; /* bits of precision in image data */

jpeg_component_info * comp_info;
/* comp_info[i] describes component that appears i'th in SOF */

boolean progressive_mode; /* TRUE if SOFn specifies progressive mode */
boolean arith_code; /* TRUE=arithmetic coding, FALSE=Huffman */

UINT8 arith_dc_L[NUM_ARITH_TBLS]; /* L values for DC arith-coding tables */
UINT8 arith_dc_U[NUM_ARITH_TBLS]; /* U values for DC arith-coding tables */
UINT8 arith_ac_K[NUM_ARITH_TBLS]; /* Kx values for AC arith-coding tables */

unsigned int restart_interval; /* MCUs per restart interval, or 0 for no restart */

/* These fields record data obtained from optional markers recognized by
* the JPEG library.
*/
boolean saw_JFIF_marker; /* TRUE iff a JFIF APP0 marker was found */
/* Data copied from JFIF marker; only valid if saw_JFIF_marker is TRUE: */
UINT8 JFIF_major_version; /* JFIF version number */
UINT8 JFIF_minor_version;

```

```

UINT8 density_unit; /* HIF code for pixel size units */
UINT16 X_density; /* Horizontal pixel density */
UINT16 Y_density; /* Vertical pixel density */
boolean saw_Adobe_marker; /* TRUE iff an Adobe APP14 marker was found */
UINT8 Adobe_transform; /* Color transform code from Adobe marker */

boolean CCIR601_sampling; /* TRUE=first samples are cosited */

/* Aside from the specific data retained from APPn markers known to the
 * library, the uninterpreted contents of any or all APPn and COM markers
 * can be saved in a list for examination by the application.
 */
jpeg_saved_marker_ptr marker_list; /* Head of list of saved markers */

/* Remaining fields are known throughout decompressor, but generally
 * should not be touched by a surrounding application.
 */

/*
 * These fields are computed during decompression startup
 */
int max_h_samp_factor; /* largest h_samp_factor */
int max_v_samp_factor; /* largest v_samp_factor */

int min_DCT_scaled_size; /* smallest DCT_scaled_size of any component */

JDIMENSION total_iMCU_rows; /* # of iMCU rows in image */
/* The coefficient controller's input and output progress is measured in
 * units of "iMCU" (interleaved MCU) rows. These are the same as MCU rows
 * in fully interleaved JPEG scans, but are used whether the scan is
 * interleaved or not. We define an iMCU row as v_samp_factor DCT block
 * rows of each component. Therefore, the IDCT output contains
 * v_samp_factor*DCT_scaled_size sample rows of a component per iMCU row.
 */
JSAMPLE * sample_range_limit; /* table for fast range-limiting */

/*
 * These fields are valid during any one scan.
 * They describe the components and MCUs actually appearing in the scan.
 * Note that the decompressor output side must not use these fields.
 */
int comps_in_scan; /* # of JPEG components in this scan */
jpeg_component_info * cur_comp_info[MAX_COMPS_IN_SCAN];
/* *cur_comp_info[i] describes component that appears i'th in SOS */

JDIMENSION MCUs_per_row; /* # of MCUs across the image */
JDIMENSION MCU_rows_in_scan; /* # of MCU rows in the image */

int blocks_in_MCU; /* # of DCT blocks per MCU */
int MCU_membership[D_MAX_BLOCKS_IN_MCU];
/* MCU_membership[i] is index in cur_comp_info of component owning */
/* i'th block in an MCU */

int Ss, Se, Ah, Al; /* progressive JPEG parameters for scan */

/* This field is shared between entropy decoder and marker parser.
 * It is either zero or the code of a JPEG marker that has been
 * read from the data source, but has not yet been processed.
 */
int unread_marker;

/*
 * Links to decompression subobjects (methods, private variables of modules)
 */
struct jpeg_decomp_master * master;
struct jpeg_d_main_controller * main;
struct jpeg_d_coef_controller * coef;
struct jpeg_d_post_controller * post;
struct jpeg_input_controller * inputctl;
struct jpeg_marker_reader * marker;
struct jpeg_entropy_decoder * entropy;
struct jpeg_inverse_dct * idct;
struct jpeg_upsampler * upsample;
struct jpeg_color_deconverter * cconvert;
struct jpeg_color_quantizer * cquantize;
};

/* "Object" declarations for JPEG modules that may be supplied or called

```

```

* directly by the surrounding application.
* As with all objects in the library, these structs only define the
* publicly visible methods and state variables of a module. Additional
* private fields may exist after the public ones.
*/

```

```

/* Error handler object */

```

```

struct jpeg_error_mgr {
    /* Error exit handler: does not return to caller */
    JMETHOD(void, error_exit, (j_common_ptr cinfo));
    /* Conditionally emit a trace or warning message */
    JMETHOD(void, emit_message, (j_common_ptr cinfo, int msg_level));
    /* Routine that actually outputs a trace or error message */
    JMETHOD(void, output_message, (j_common_ptr cinfo));
    /* Format a message string for the most recent JPEG error or message */
    JMETHOD(void, format_message, (j_common_ptr cinfo, char * buffer));
#define JMSG_LENGTH_MAX 200 /* recommended size of format_message buffer */
    /* Reset error state variables at start of a new image */
    JMETHOD(void, reset_error_mgr, (j_common_ptr cinfo));

```

```

    /* The message ID code and any parameters are saved here.
     * A message can have one string parameter or up to 8 int parameters.
     */

```

```

    int msg_code;
#define JMSG_STR_PARM_MAX 80
    union {
        int i[8];
        char s[JMSG_STR_PARM_MAX];
    } msg_parm;

```

```

/* Standard state variables for error facility */

```

```

int trace_level; /* max msg_level that will be displayed */

```

```

/* For recoverable corrupt-data errors, we emit a warning message,
 * but keep going unless emit_message chooses to abort. emit_message
 * should count warnings in num_warnings. The surrounding application
 * can check for bad data by seeing if num_warnings is nonzero at the
 * end of processing.
 */

```

```

long num_warnings; /* number of corrupt-data warnings */

```

```

/* These fields point to the table(s) of error message strings.
 * An application can change the table pointer to switch to a different
 * message list (typically, to change the language in which errors are
 * reported). Some applications may wish to add additional error codes
 * that will be handled by the JPEG library error mechanism; the second
 * table pointer is used for this purpose.
 */

```

```

/* First table includes all errors generated by JPEG library itself.
 * Error code 0 is reserved for a "no such error string" message.
 */

```

```

const char * const * jpeg_message_table; /* Library errors */
int last_jpeg_message; /* Table contains strings 0..last_jpeg_message */
/* Second table can be added by application (see cjpeg/djpeg for example).
 * It contains strings numbered first_addon_message..last_addon_message.
 */

```

```

const char * const * addon_message_table; /* Non-library errors */
int first_addon_message; /* code for first string in addon table */
int last_addon_message; /* code for last string in addon table */

```

```

};

```

```

/* Progress monitor object */

```

```

struct jpeg_progress_mgr {
    JMETHOD(void, progress_monitor, (j_common_ptr cinfo));

    long pass_counter; /* work units completed in this pass */
    long pass_limit; /* total number of work units in this pass */
    int completed_passes; /* passes completed so far */
    int total_passes; /* total number of passes expected */
};

```

```

/* Data destination object for compression */

```

```

/* ### Imtiaz: The linked list structure to store the buffer data. */

```

```

typedef struct linked_list
{
    JOCTET *buffer;
    struct linked_list *next;
} buffer_list;

struct jpeg_destination_mgr {
    JOCTET * next_output_byte; /* => next byte to write in buffer */
    size_t free_in_buffer; /* # of byte spaces remaining in buffer */

    JOCTET *outbuffer;

    buffer_list *head_ptr;
    buffer_list *current_ptr;

    JMETHOD(void, init_destination, (j_compress_ptr cinfo));
    JMETHOD(boolean, empty_output_buffer, (j_compress_ptr cinfo));
    JMETHOD(void, term_destination, (j_compress_ptr cinfo));
};

/* Data source object for decompression */

struct jpeg_source_mgr {
    const JOCTET * next_input_byte; /* => next byte to read from buffer */
    size_t bytes_in_buffer; /* # of bytes remaining in buffer */

    JSAMPLE *inbuffer; /* ### Imtiaz: I added this */
    JSAMPLE *head_inbuffer; /* " */

    int buffer_length;

    JMETHOD(void, init_source, (j_decompress_ptr cinfo));
    JMETHOD(boolean, fill_input_buffer, (j_decompress_ptr cinfo));
    JMETHOD(void, skip_input_data, (j_decompress_ptr cinfo, long num_bytes));
    JMETHOD(boolean, resync_to_restart, (j_decompress_ptr cinfo, int desired));
    JMETHOD(void, term_source, (j_decompress_ptr cinfo));
}

/*
 * Memory manager object.
 * Allocates "small" objects (a few K total), "large" objects (tens of K),
 * and "really big" objects (virtual arrays with backing store if needed).
 * The memory manager does not allow individual objects to be freed; rather,
 * each created object is assigned to a pool, and whole pools can be freed
 * at once. This is faster and more convenient than remembering exactly what
 * to free, especially where malloc()/free() are not too speedy.
 * NB: alloc routines never return NULL. They exit to error_exit if not
 * successful.
 */

#define JPOOL_PERMANENT 0 /* lasts until master record is destroyed */
#define JPOOL_IMAGE 1 /* lasts until done with image/datastream */
#define JPOOL_NUMPOOLS 2

typedef struct jvirt_sarray_control * jvirt_sarray_ptr;
typedef struct jvirt_barray_control * jvirt_barray_ptr;

struct jpeg_memory_mgr {
    /* Method pointers */
    JMETHOD(void *, alloc_small, (j_common_ptr cinfo, int pool_id,
        size_t sizeofobject));
    JMETHOD(void *, alloc_large, (j_common_ptr cinfo, int pool_id,
        size_t sizeofobject));
    JMETHOD(JSAMPARRAY, alloc_sarray, (j_common_ptr cinfo, int pool_id,
        JDIMENSION samplesperrow,
        JDIMENSION numrows));
    JMETHOD(JBLOCKARRAY, alloc_barray, (j_common_ptr cinfo, int pool_id,
        JDIMENSION blocksperrow,
        JDIMENSION numrows));
    JMETHOD(jvirt_sarray_ptr, request_virt_sarray, (j_common_ptr cinfo,
        int pool_id,
        boolean pre_zero,
        JDIMENSION samplesperrow,
        JDIMENSION numrows,

```

```

        JDIMENSION maxaccess));
JMETHOD(jvirt_barray_ptr, realize_virt_barray, (j_common_ptr cinfo,
        int pool_id,
        boolean pre_zero,
        JDIMENSION blocksperrow,
        JDIMENSION numrows,
        JDIMENSION maxaccess));
JMETHOD(void, realize_virt_arrays, (j_common_ptr cinfo));
JMETHOD(JSAMPARRAY, access_virt_sarray, (j_common_ptr cinfo,
        jvirt_sarray_ptr ptr,
        JDIMENSION start_row,
        JDIMENSION num_rows,
        boolean writable));
JMETHOD(JBLOCKARRAY, access_virt_barray, (j_common_ptr cinfo,
        jvirt_barray_ptr ptr,
        JDIMENSION start_row,
        JDIMENSION num_rows,
        boolean writable));
JMETHOD(void, free_pool, (j_common_ptr cinfo, int pool_id));
JMETHOD(void, self_destruct, (j_common_ptr cinfo));

/* Limit on memory allocation for this JPEG object.  (Note that this is
 * merely advisory, not a guaranteed maximum; it only affects the space
 * used for virtual-array buffers.)  May be changed by outer application
 * after creating the JPEG object.
 */
long max_memory_to_use;

/* Maximum allocation request accepted by alloc_large. */
long max_alloc_chunk;
};

/* Routine signature for application-supplied marker processing methods.
 * Need not pass marker code since it is stored in cinfo->unread_marker.
 */
typedef JMETHOD(boolean, jpeg_marker_parser_method, (j_decompress_ptr cinfo));

/* Declarations for routines called by application.
 * The JPP macro hides prototype parameters from compilers that can't cope.
 * Note JPP requires double parentheses.
 */
#ifdef HAVE_PROTOTYPES
#define JPP(arglist)    arglist
#else
#define JPP(arglist)    ()
#endif

/* Short forms of external names for systems with brain-damaged linkers.
 * We shorten external names to be unique in the first six letters, which
 * is good enough for all known systems.
 * (If your compiler itself needs names to be unique in less than 15
 * characters, you are out of luck.  Get a better compiler.)
 */
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_std_error        jStdError
#define jpeg_CreateCompress  jCreaCompress
#define jpeg_CreateDecompress jCreaDecompress
#define jpeg_destroy_compress jDestCompress
#define jpeg_destroy_decompress jDestDecompress
#define jpeg_stdio_dest      jStdDest
#define jpeg_buffer_dest     jBufDest
#define stitch_list          S_List
#define jpeg_stdio_src       jStdSrc
#define jpeg_set_defaults    jSetDefaults
#define jpeg_set_colorspace  jSetColorspace
#define jpeg_default_colorspace jDefColorspace
#define jpeg_set_quality      jSetQuality
#define jpeg_set_linear_quality jSetLQuality
#define jpeg_add_quant_table  jAddQuantTable
#define jpeg_quality_scaling  jQualityScaling
#define jpeg_simple_progression jSimProgress
#define jpeg_suppress_tables  jSuppressTables
#define jpeg_alloc_quant_table jAlcQTable
#define jpeg_alloc_huff_table jAlcHTable
#define jpeg_start_compress  jStrtCompress

```



```

#define jpeg_write_scanlines jWrtScanlines
#define jpeg_finish_compress jFinCompress
#define jpeg_write_raw_data jWrtRawData
#define jpeg_write_marker jWrtMarker
#define jpeg_write_m_header jWrtMHeader
#define jpeg_write_m_byte jWrtMByte
#define jpeg_write_tables jWrtTables
#define jpeg_read_header jReadHeader
#define jpeg_start_decompress jStrtDecompress
#define jpeg_read_scanlines jReadScanlines
#define jpeg_finish_decompress jFinDecompress
#define jpeg_read_raw_data jReadRawData
#define jpeg_has_multiple_scans jHasMultScn
#define jpeg_start_output jStrtOutput
#define jpeg_finish_output jFinOutput
#define jpeg_input_complete jInComplete
#define jpeg_new_colormap jNewCMap
#define jpeg_consume_input jConsumeInput
#define jpeg_calc_output_dimensions jCalcDimensions
#define jpeg_save_markers jSaveMarkers
#define jpeg_set_marker_processor jSetMarker
#define jpeg_read_coefficients jReadCoefs
#define jpeg_write_coefficients jWrtCoefs
#define jpeg_copy_critical_parameters jCopyCrit
#define jpeg_abort_compress jAbrtCompress
#define jpeg_abort_decompress jAbrtDecompress
#define jpeg_abort jAbort
#define jpeg_destroy jDestroy
#define jpeg_resync_to_restart jResyncRestart
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Default error-management setup */
EXTERN(struct jpeg_error_mgr *) jpeg_std_error
    JPP((struct jpeg_error_mgr * err));

/* Initialization of JPEG compression objects.
 * jpeg_create_compress() and jpeg_create_decompress() are the exported
 * names that applications should call. These expand to calls on
 * jpeg_CreateCompress and jpeg_CreateDecompress with additional information
 * passed for version mismatch checking.
 * NB: you must set up the error-manager BEFORE calling jpeg_create_xxx.
 */
#define jpeg_create_compress(cinfo) \
    jpeg_CreateCompress((cinfo), JPEG_LIB_VERSION, \
        (size_t) sizeof(struct jpeg_compress_struct))
#define jpeg_create_decompress(cinfo) \
    jpeg_CreateDecompress((cinfo), JPEG_LIB_VERSION, \
        (size_t) sizeof(struct jpeg_decompress_struct))
EXTERN(void) jpeg_CreateCompress JPP((j_compress_ptr cinfo,
    int version, size_t structsize));
EXTERN(void) jpeg_CreateDecompress JPP((j_decompress_ptr cinfo,
    int version, size_t structsize));
/* Destruction of JPEG compression objects */
EXTERN(void) jpeg_destroy_compress JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_destroy_decompress JPP((j_decompress_ptr cinfo));

/* Standard data source and destination managers: stdio streams. */
/* Caller is responsible for opening the file before and closing after. */

EXTERN(void) jpeg_stdio_dest JPP((j_compress_ptr cinfo, FILE * outfile));

EXTERN(void) jpeg_buffer_dest JPP((j_compress_ptr cinfo));
EXTERN(void) stitch_list JPP((j_compress_ptr cinfo));

EXTERN(void) jpeg_stdio_src JPP((j_decompress_ptr cinfo, FILE * infile));
EXTERN(void) jpeg_buffer_src JPP((j_decompress_ptr cinfo));

/* Default parameter setup for compression */
EXTERN(void) jpeg_set_defaults JPP((j_compress_ptr cinfo));
/* Compression parameter setup aids */
EXTERN(void) jpeg_set_colorspace JPP((j_compress_ptr cinfo,
    J_COLOR_SPACE colorspace));
EXTERN(void) jpeg_default_colorspace JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_set_quality JPP((j_compress_ptr cinfo, int quality,
    boolean force_baseline));
EXTERN(void) jpeg_set_linear_quality JPP((j_compress_ptr cinfo,
    int scale_factor,

```

```

        boolean force_baseline));
EXTERN(void) jpeg_add_quant_tbl JPP((j_compress_ptr cinfo, int tbl,
        const unsigned int *basic_table,
        int scale_factor,
        boolean force_baseline));
EXTERN(int) jpeg_quality_scaling JPP((int quality));
EXTERN(void) jpeg_simple_progression JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_suppress_tables JPP((j_compress_ptr cinfo,
        boolean suppress));
EXTERN(JQUANT_TBL *) jpeg_alloc_quant_table JPP((j_common_ptr cinfo));
EXTERN(JHUFF_TBL *) jpeg_alloc_huff_table JPP((j_common_ptr cinfo));

/* Main entry points for compression */
EXTERN(void) jpeg_start_compress JPP((j_compress_ptr cinfo,
        boolean write_all_tables));
EXTERN(JDIMENSION) jpeg_write_scanlines JPP((j_compress_ptr cinfo,
        JSAMPARRAY scanlines,
        JDIMENSION num_lines));
EXTERN(void) jpeg_finish_compress JPP((j_compress_ptr cinfo));

/* Replaces jpeg_write_scanlines when writing raw downsampled data. */
EXTERN(JDIMENSION) jpeg_write_raw_data JPP((j_compress_ptr cinfo,
        JSAMPIMAGE data,
        JDIMENSION num_lines));

/* Write a special marker. See libjpeg.doc concerning safe usage. */
EXTERN(void) jpeg_write_marker
        JPP((j_compress_ptr cinfo, int marker,
        const JOCTET *dataptr, unsigned int datalen));
/* Same, but piecemeal. */
EXTERN(void) jpeg_write_m_header
        JPP((j_compress_ptr cinfo, int marker, unsigned int datalen));
EXTERN(void) jpeg_write_m_byte
        JPP((j_compress_ptr cinfo, int val));

/* Alternate compression function: just write an abbreviated table file */
EXTERN(void) jpeg_write_tables JPP((j_compress_ptr cinfo));

/* Decompression startup: read start of JPEG datastream to see what's there */
EXTERN(int) jpeg_read_header JPP((j_decompress_ptr cinfo,
        boolean require_image));
/* Return value is one of: */
#define JPEG_SUSPENDED 0 /* Suspended due to lack of input data */
#define JPEG_HEADER_OK 1 /* Found valid image datastream */
#define JPEG_HEADER_TABLES_ONLY 2 /* Found valid table-specs-only datastream */
/* If you pass require_image = TRUE (normal case), you need not check for
 * a TABLES_ONLY return code; an abbreviated file will cause an error exit.
 * JPEG_SUSPENDED is only possible if you use a data source module that can
 * give a suspension return (the stdio source module doesn't).
 */

/* Main entry points for decompression */
EXTERN(boolean) jpeg_start_decompress JPP((j_decompress_ptr cinfo));
EXTERN(JDIMENSION) jpeg_read_scanlines JPP((j_decompress_ptr cinfo,
        JSAMPARRAY scanlines,
        JDIMENSION max_lines));
EXTERN(boolean) jpeg_finish_decompress JPP((j_decompress_ptr cinfo));

/* Replaces jpeg_read_scanlines when reading raw downsampled data. */
EXTERN(JDIMENSION) jpeg_read_raw_data JPP((j_decompress_ptr cinfo,
        JSAMPIMAGE data,
        JDIMENSION max_lines));

/* Additional entry points for buffered-image mode. */
EXTERN(boolean) jpeg_has_multiple_scans JPP((j_decompress_ptr cinfo));
EXTERN(boolean) jpeg_start_output JPP((j_decompress_ptr cinfo,
        int scan_number));
EXTERN(boolean) jpeg_finish_output JPP((j_decompress_ptr cinfo));
EXTERN(boolean) jpeg_input_complete JPP((j_decompress_ptr cinfo));
EXTERN(void) jpeg_new_colormap JPP((j_decompress_ptr cinfo));
EXTERN(int) jpeg_consume_input JPP((j_decompress_ptr cinfo));
/* Return value is one of: */
/* #define JPEG_SUSPENDED 0 Suspended due to lack of input data */
#define JPEG_REACHED_SOS 1 /* Reached start of new scan */
#define JPEG_REACHED_EOI 2 /* Reached end of image */
#define JPEG_ROW_COMPLETED 3 /* Completed one iMCU row */
#define JPEG_SCAN_COMPLETED 4 /* Completed last iMCU row of a scan */

/* Precalculate output dimensions for current decompression parameters. */
EXTERN(void) jpeg_calc_output_dimensions JPP((j_decompress_ptr cinfo));

```

```

/* Control saving of COM and APPn markers into marker_list. */
EXTERN(void) jpeg_save_markers JPP((j_decompress_ptr cinfo, int marker_code,
    unsigned int length_limit));

/* Install a special processing method for COM or APPn markers. */
EXTERN(void) jpeg_set_marker_processor JPP((j_decompress_ptr cinfo, int marker_code,
    jpeg_marker_parser_method routine));

/* Read or write raw DCT coefficients --- useful for lossless transcoding. */
EXTERN(jvirt_barray_ptr *) jpeg_read_coefficients JPP((j_decompress_ptr cinfo));
EXTERN(void) jpeg_write_coefficients JPP((j_compress_ptr cinfo,
    jvirt_barray_ptr * coef_arrays));
EXTERN(void) jpeg_copy_critical_parameters JPP((j_decompress_ptr srcinfo,
    j_compress_ptr dstinfo));

/* If you choose to abort compression or decompression before completing
 * jpeg_finish_(de)compress, then you need to clean up to release memory,
 * temporary files, etc. You can just call jpeg_destroy_(de)compress
 * if you're done with the JPEG object, but if you want to clean it up and
 * reuse it, call this:
 */
EXTERN(void) jpeg_abort_compress JPP((j_compress_ptr cinfo));
EXTERN(void) jpeg_abort_decompress JPP((j_decompress_ptr cinfo));

/* Generic versions of jpeg_abort and jpeg_destroy that work on either
 * flavor of JPEG object. These may be more convenient in some places.
 */
EXTERN(void) jpeg_abort JPP((j_common_ptr cinfo));
EXTERN(void) jpeg_destroy JPP((j_common_ptr cinfo));

/* Default restart-marker-resync procedure for use by data source modules */
EXTERN(boolean) jpeg_resync_to_restart JPP((j_decompress_ptr cinfo,
    int desired));

/* These marker codes are exported since applications and data source modules
 * are likely to want to use them.
 */
#define JPEG_RST0 0xD0 /* RST0 marker code */
#define JPEG_EOI 0xD9 /* EOI marker code */
#define JPEG_APP0 0xE0 /* APP0 marker code */
#define JPEG_COM 0xFE /* COM marker code */

/* If we have a brain-damaged compiler that emits warnings (or worse, errors)
 * for structure definitions that are never filled in, keep it quiet by
 * supplying dummy definitions for the various substructures.
 */
#ifdef INCOMPLETE_TYPES_BROKEN
#ifdef JPEG_INTERNALS /* will be defined in jpegint.h */
struct jvirt_sarray_control { long dummy; };
struct jvirt_barray_control { long dummy; };
struct jpeg_comp_master { long dummy; };
struct jpeg_c_main_controller { long dummy; };
struct jpeg_c_prep_controller { long dummy; };
struct jpeg_c_coef_controller { long dummy; };
struct jpeg_marker_writer { long dummy; };
struct jpeg_color_converter { long dummy; };
struct jpeg_downsampler { long dummy; };
struct jpeg_forward_dct { long dummy; };
struct jpeg_entropy_encoder { long dummy; };
struct jpeg_decomp_master { long dummy; };
struct jpeg_d_main_controller { long dummy; };
struct jpeg_d_coef_controller { long dummy; };
struct jpeg_d_post_controller { long dummy; };
struct jpeg_input_controller { long dummy; };
struct jpeg_marker_reader { long dummy; };
struct jpeg_entropy_decoder { long dummy; };
struct jpeg_inverse_dct { long dummy; };
struct jpeg_upsampler { long dummy; };
struct jpeg_color_deconverter { long dummy; };
struct jpeg_color_quantizer { long dummy; };
#endif /* JPEG_INTERNALS */
#endif /* INCOMPLETE_TYPES_BROKEN */

```

```

/*
 * The JPEG library modules define JPEG_INTERNALS before including this file.
 * The internal structure declarations are read only when that is true.
 * Applications using the library should not include jpegint.h, but may wish
 * to include jerror.h.
 */

```

```
#ifndef JPEG_INTERNALS
#include "jpegint.h"          /* fetch private declarations */
#include "jerror.h"           /* fetch error codes too */
#endif

#endif /* JPEGLIB_H */
```

1. *Staphylococcus aureus* (Staph. aureus) is a common cause of skin infections, such as abscesses and boils. It is also responsible for food poisoning and hospital-acquired infections.



```

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////

RawPixelEncoder::RawPixelEncoder()
{
    m_ReleaseBufferMemory=false;
    m_pBuffer=NULL;
    m_BufferPtr=0;
}
RawPixelEncoder::~RawPixelEncoder() {    ReleaseBuffer();    }

/*****
*
*   Prepare buffer for data entry
*
*****/
bool RawPixelEncoder::SetSize(UINT32 size)
{
    ReleaseBuffer();
    if(size<=0) return false;
    m_Size=size;
    try
    {
        m_pBuffer = new BYTE[m_Size];
    }
    catch(...) {    return false;    }
    if(m_pBuffer)
    {
        memset(m_pBuffer,0,m_Size);
        m_ReleaseBufferMemory=true;
        m_BufferPtr=0;
        return true;
    }
    else    return false;
}

/*****
*
*   Add new subbufer
*
*****/
UINT32 RawPixelEncoder::AddData(BYTE* buf, UINT32 buf_size,
                                UINT32 fliprawbytes /*=0*/)
{
    bool flipY = (fliprawbytes>1);
    if(m_BufferPtr>=m_Size) return 0;
    if(buf_size>m_Size-m_BufferPtr) // avoid overflow
    {
        flipY=false;
        buf_size = m_Size-m_BufferPtr;
    }
    BYTE *start = &(m_pBuffer[m_BufferPtr]);
    for(UINT32 k=0; k<buf_size; k++)
    {
        m_pBuffer[m_BufferPtr] = buf[k];
        m_BufferPtr++;
    }
    if(flipY)
    {
        ::Flip2DBufferY(start, buf_size, fliprawbytes);
    }
    start=NULL;
    return buf_size;
}

/*****
*
*   Transfer buffer data to VR
*
*****/
void RawPixelEncoder::TransferDataToVR(VR *vr)
{
    vr->AttachData(m_pBuffer,m_Size);
    m_pBuffer=NULL;
    m_ReleaseBufferMemory=FALSE;
}
/*****
*
*/

```

```
*   Release allocated buffer memory
*
*****
void RawPixelEncoder::ReleaseBuffer()
{
    if(m_ReleaseBufferMemory)
    {
        if(m_pBuffer)
        {
            delete [] m_pBuffer;
            m_pBuffer=NULL;
        }
        m_ReleaseBufferMemory=false;
    }
}
```

1. **Introduction**  
 2. **Background**  
 3. **Methodology**  
 4. **Results**  
 5. **Discussion**  
 6. **Conclusion**  
 7. **References**  
 8. **Appendix**  
 9. **Notes**  
 10. **References**  
 11. **Appendix**  
 12. **Notes**  
 13. **References**  
 14. **Appendix**  
 15. **Notes**  
 16. **References**  
 17. **Appendix**  
 18. **Notes**  
 19. **References**  
 20. **Appendix**  
 21. **Notes**  
 22. **References**  
 23. **Appendix**  
 24. **Notes**  
 25. **References**  
 26. **Appendix**  
 27. **Notes**  
 28. **References**  
 29. **Appendix**  
 30. **Notes**  
 31. **References**  
 32. **Appendix**  
 33. **Notes**  
 34. **References**  
 35. **Appendix**  
 36. **Notes**  
 37. **References**  
 38. **Appendix**  
 39. **Notes**  
 40. **References**  
 41. **Appendix**  
 42. **Notes**  
 43. **References**  
 44. **Appendix**  
 45. **Notes**  
 46. **References**  
 47. **Appendix**  
 48. **Notes**  
 49. **References**  
 50. **Appendix**  
 51. **Notes**  
 52. **References**  
 53. **Appendix**  
 54. **Notes**  
 55. **References**  
 56. **Appendix**  
 57. **Notes**  
 58. **References**  
 59. **Appendix**  
 60. **Notes**  
 61. **References**  
 62. **Appendix**  
 63. **Notes**  
 64. **References**  
 65. **Appendix**  
 66. **Notes**  
 67. **References**  
 68. **Appendix**  
 69. **Notes**  
 70. **References**  
 71. **Appendix**  
 72. **Notes**  
 73. **References**  
 74. **Appendix**  
 75. **Notes**  
 76. **References**  
 77. **Appendix**  
 78. **Notes**  
 79. **References**  
 80. **Appendix**  
 81. **Notes**  
 82. **References**  
 83. **Appendix**  
 84. **Notes**  
 85. **References**  
 86. **Appendix**  
 87. **Notes**  
 88. **References**  
 89. **Appendix**  
 90. **Notes**  
 91. **References**  
 92. **Appendix**  
 93. **Notes**  
 94. **References**  
 95. **Appendix**  
 96. **Notes**  
 97. **References**  
 98. **Appendix**  
 99. **Notes**  
 100. **References**

```

/*
 * jchuff.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains declarations for Huffman entropy encoding routines
 * that are shared between the sequential encoder (jchuff.c) and the
 * progressive encoder (jcphuff.c).  No other modules need to see these.
 */

/* The legal range of a DCT coefficient is
 * -1024 .. +1023 for 8-bit data;
 * -16384 .. +16383 for 12-bit data.
 * Hence the magnitude should always fit in 10 or 14 bits respectively.
 */

#if BITS_IN_JSAMPLE == 8
#define MAX_COEF_BITS 10
#else
#define MAX_COEF_BITS 14
#endif

/* Derived data constructed for each Huffman table */
typedef struct {
  unsigned int ehufco[256]; /* code for each symbol */
  char ehufsi[256]; /* length of code for each symbol */
  /* If no code has been allocated for a symbol S, ehufsi[S] contains 0 */
} c_derived_tbl;

/* Short forms of external names for systems with brain-damaged linkers. */
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_make_c_derived_tbl jMkCDerived
#define jpeg_gen_optimal_table jGenOptTbl
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Expand a Huffman table definition into the derived format */
EXTERN(void) jpeg_make_c_derived_tbl
  JPP((j_compress_ptr cinfo, boolean isDC, int tblno,
       c_derived_tbl ** pdtbl));

/* Generate an optimal table definition given the specified counts */
EXTERN(void) jpeg_gen_optimal_table
  JPP((j_compress_ptr cinfo, JHUFF_TBL * htbl, long freq[]));

```



```

/* jconfig.vc --- jconfig.h for Microsoft Visual C++ on Windows 95 NT. */
/* see jconfig.doc for explanations */

```

```

#define HAVE_PROTOTYPES
#define HAVE_UNSIGNED_CHAR
#define HAVE_UNSIGNED_SHORT
/* #define void char */
/* #define const */
#undef CHAR_IS_UNSIGNED
#define HAVE_STDDEF_H
#define HAVE_STDLIB_H
#undef NEED_BSD_STRINGS
#undef NEED_SYS_TYPES_H
#undef NEED_FAR_POINTERS /* we presume a 32-bit flat memory model */
#undef NEED_SHORT_EXTERNAL_NAMES
#undef INCOMPLETE_TYPES_BROKEN

```

```

/* Define "boolean" as unsigned char, not int, per Windows custom */
#ifdef __RPCNDR_H /* don't conflict if rpcnldr.h already read */
typedef unsigned char boolean;
#endif
#define HAVE_BOOLEAN /* prevent jmorecfg.h from redefining it */

```

```

#ifdef JPEG_INTERNALS
#undef RIGHT_SHIFT_IS_UNSIGNED

#endif /* JPEG_INTERNALS */

```

```

#ifdef JPEG_CJPEG_DJPEG
#define BMP_SUPPORTED /* BMP image file format */
#define GIF_SUPPORTED /* GIF image file format */
#define PPM_SUPPORTED /* PBMPLUS PPM/PGM image file format */
#undef RLE_SUPPORTED /* Utah RLE image file format */
#define TARGA_SUPPORTED /* Targa image file format */

#define TWO_FILE_COMMANDLINE /* optional */
#define USE_SETMODE /* Microsoft has setmode() */
#undef NEED_SIGNAL_CATCHER
#undef DONT_USE_B_MODE
#undef PROGRESS_REPORT /* optional */

#endif /* JPEG_CJPEG_DJPEG */

```

```

/*
 * jdct.h
 *
 * Copyright (C) 1994-1996, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This include file contains common declarations for the forward and
 * inverse DCT modules. These declarations are private to the DCT managers
 * (jcdctmgr.c, jddctmgr.c) and the individual DCT algorithms.
 * The individual DCT algorithms are kept in separate files to ease
 * machine-dependent tuning (e.g., assembly coding).
 */

/*
 * A forward DCT routine is given a pointer to a work area of type DCTELEM[];
 * the DCT is to be performed in-place in that buffer. Type DCTELEM is int
 * for 8-bit samples, INT32 for 12-bit samples. (NOTE: Floating-point DCT
 * implementations use an array of type FAST_FLOAT, instead.)
 * The DCT inputs are expected to be signed (range +/-CENTERJSAMPLE).
 * The DCT outputs are returned scaled up by a factor of 8; they therefore
 * have a range of +/-8K for 8-bit data, +/-128K for 12-bit data. This
 * convention improves accuracy in integer implementations and saves some
 * work in floating-point ones.
 * Quantization of the output coefficients is done by jcdctmgr.c.
 */

#if BITS_IN_JSAMPLE == 8
typedef int DCTELEM;          /* 16 or 32 bits is fine */
#else
typedef INT32 DCTELEM;       /* must have 32 bits */
#endif

typedef JMETHOD(void, forward_DCT_method_ptr, (DCTELEM * data));
typedef JMETHOD(void, float_DCT_method_ptr, (FAST_FLOAT * data));

/*
 * An inverse DCT routine is given a pointer to the input JBLOCK and a pointer
 * to an output sample array. The routine must dequantize the input data as
 * well as perform the IDCT; for dequantization, it uses the multiplier table
 * pointed to by compptr->dct_table. The output data is to be placed into the
 * sample array starting at a specified column. (Any row offset needed will
 * be applied to the array pointer before it is passed to the IDCT code.)
 * Note that the number of samples emitted by the IDCT routine is
 * DCT_scaled_size * DCT_scaled_size.
 */

/*typedef inverse_DCT_method_ptr is declared in jpegint.h */

/*
 * Each IDCT routine has its own ideas about the best dct_table element type.
 */

typedef MULTIPLIER ISLOW_MULT_TYPE; /* short or int, whichever is faster */
#if BITS_IN_JSAMPLE == 8
typedef MULTIPLIER IFAST_MULT_TYPE; /* 16 bits is OK, use short if faster */
#define IFAST_SCALE_BITS 2 /* fractional bits in scale factors */
#else
typedef INT32 IFAST_MULT_TYPE; /* need 32 bits for scaled quantizers */
#define IFAST_SCALE_BITS 13 /* fractional bits in scale factors */
#endif
typedef FAST_FLOAT FLOAT_MULT_TYPE; /* preferred floating type */

/*
 * Each IDCT routine is responsible for range-limiting its results and
 * converting them to unsigned form (0..MAXJSAMPLE). The raw outputs could
 * be quite far out of range if the input data is corrupt, so a bulletproof
 * range-limiting step is required. We use a mask-and-table-lookup method
 * to do the combined operations quickly. See the comments with
 * prepare_range_limit_table (in jdmaster.c) for more info.
 */

#define IDCT_range_limit(cinfo) ((cinfo)->sample_range_limit + CENTERJSAMPLE)
#define RANGE_MASK (MAXJSAMPLE * 4 + 3) /* 2 bits wider than legal samples */

```

```
/* Short forms of external names for systems with brain-damaged linkers. */
```

```
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_fdct_islow    jFDIslow
#define jpeg_fdct_ifast    jFDIfast
#define jpeg_fdct_float    jFDfloat
#define jpeg_idct_islow    jRDislow
#define jpeg_idct_ifast    jRDIfast
#define jpeg_idct_float    jRDfloat
#define jpeg_idct_4x4      jRD4x4
#define jpeg_idct_2x2      jRD2x2
#define jpeg_idct_1x1      jRD1x1
#endif /* NEED_SHORT_EXTERNAL_NAMES */
```

```
/* Extern declarations for the forward and inverse DCT routines. */
```

```
EXTERN(void) jpeg_fdct_islow JPP((DCTELEM * data));
EXTERN(void) jpeg_fdct_ifast JPP((DCTELEM * data));
EXTERN(void) jpeg_fdct_float JPP((FAST_FLOAT * data));
```

```
EXTERN(void) jpeg_idct_islow
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_ifast
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_float
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_4x4
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_2x2
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
EXTERN(void) jpeg_idct_1x1
    JPP((j_decompress_ptr cinfo, jpeg_component_info * compptr,
        JOEFPTR coef_block, JSAMPARRAY output_buf, JDIMENSION output_col));
```

```
/* Macros for handling fixed-point arithmetic; these are used by many
but not all of the DCT/IDCT modules.
*/
```

```
/* All values are expected to be of type INT32.
* Fractional constants are scaled left by CONST_BITS bits.
* CONST_BITS is defined within each module using these macros,
and may differ from one module to the next.
*/
```

```
#define ONE ((INT32) 1)
#define CONST_SCALE (ONE << CONST_BITS)
```

```
/* Convert a positive real constant to an integer scaled by CONST_SCALE.
* Caution: some C compilers fail to reduce "FIX(constant)" at compile time,
* thus causing a lot of useless floating-point operations at run time.
*/
```

```
#define FIX(x) ((INT32) ((x) * CONST_SCALE + 0.5))
```

```
/* Descale and correctly round an INT32 value that's scaled by N bits.
* We assume RIGHT_SHIFT rounds towards minus infinity, so adding
* the fudge factor is correct for either sign of X.
*/
```

```
#define DESCALE(x,n) RIGHT_SHIFT((x) + (ONE << ((n)-1)), n)
```

```
/* Multiply an INT32 variable by an INT32 constant to yield an INT32 result.
* This macro is used only when the two inputs will actually be no more than
* 16 bits wide, so that a 16x16->32 bit multiply can be used instead of a
* full 32x32 multiply. This provides a useful speedup on many machines.
* Unfortunately there is no way to specify a 16x16->32 multiply portably
* in C, but some C compilers will do the right thing if you provide the
* correct combination of casts.
*/
```

```
#ifdef SHORTxSHORT_32 /* may work if 'int' is 32 bits */
#define MULTIPLY16C16(var,const) (((INT16) (var)) * ((INT16) (const)))
#endif
#ifdef SHORTxLCONST_32 /* known to work with Microsoft C 6.0 */
```

```
#define MULTIPLY16C16(var,const) (((INT16) (var)) * ((INT32) (const)))
#endif

#ifndef MULTIPLY16C16 /* default definition */
#define MULTIPLY16C16(var,const) ((var) * (const))
#endif

/* Same except both inputs are variables. */

#ifdef SHORTxSHORT_32 /* may work if 'int' is 32 bits */
#define MULTIPLY16V16(var1,var2) (((INT16) (var1)) * ((INT16) (var2)))
#endif

#ifndef MULTIPLY16V16 /* default definition */
#define MULTIPLY16V16(var1,var2) ((var1) * (var2))
#endif
```

[illegible]

```

/*
 * jdhuft.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains declarations for Huffman entropy decoding routines
 * that are shared between the sequential decoder (jdhuff.c) and the
 * progressive decoder (jdp Huff.c). No other modules need to see these.
 */

/* Short forms of external names for systems with brain-damaged linkers. */

#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_make_d_derived_tbl jMkDDerived
#define jpeg_fill_bit_buffer jFilBitBuf
#define jpeg_huff_decode jHufDecode
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/* Derived data constructed for each Huffman table */
#define HUFF_LOOKAHEAD 8 /* # of bits of lookahead */

typedef struct {
  /* Basic tables: (element [0] of each array is unused) */
  INT32 maxcode[18]; /* largest code of length k (-1 if none) */
  /* (maxcode[17] is a sentinel to ensure jpeg_huff_decode terminates) */
  INT32 valoffset[17]; /* huffval[] offset for codes of length k */
  /* valoffset[k] = huffval[] index of 1st symbol of code length k, less
   * the smallest code of length k; so given a code of length k, the
   * corresponding symbol is huffval[code + valoffset[k]]
   */
  /* Link to public Huffman table (needed only in jpeg_huff_decode) */
  const HUFF_TBL *pub;

  /* Lookahead tables: indexed by the next HUFF_LOOKAHEAD bits of
   * the input data stream. If the next Huffman code is no more
   * than HUFF_LOOKAHEAD bits long, we can obtain its length and
   * the corresponding symbol directly from these tables.
   */
  int look_nbits[1<<HUFF_LOOKAHEAD]; /* # bits, or 0 if too long */
  UINT8 look_sym[1<<HUFF_LOOKAHEAD]; /* symbol, or unused */
} d_derived_tbl;

/* Expand a Huffman table definition into the derived format */
EXTERN(void) jpeg_make_d_derived_tbl
  JPP((j_decompress_ptr cinfo, boolean isDC, int tblno,
       d_derived_tbl ** pdtbl));

/*
 * Fetching the next N bits from the input stream is a time-critical operation
 * for the Huffman decoders. We implement it with a combination of inline
 * macros and out-of-line subroutines. Note that N (the number of bits
 * demanded at one time) never exceeds 15 for JPEG use.
 *
 * We read source bytes into get_buffer and dole out bits as needed.
 * If get_buffer already contains enough bits, they are fetched in-line
 * by the macros CHECK_BIT_BUFFER and GET_BITS. When there aren't enough
 * bits, jpeg_fill_bit_buffer is called; it will attempt to fill get_buffer
 * as full as possible (not just to the number of bits needed; this
 * prefetching reduces the overhead cost of calling jpeg_fill_bit_buffer).
 * Note that jpeg_fill_bit_buffer may return FALSE to indicate suspension.
 * On TRUE return, jpeg_fill_bit_buffer guarantees that get_buffer contains
 * at least the requested number of bits --- dummy zeroes are inserted if
 * necessary.
 */

typedef INT32 bit_buf_type; /* type of bit-extraction buffer */
#define BIT_BUF_SIZE 32 /* size of buffer in bits */

/* If long is > 32 bits on your machine, and shifting/masking longs is
 * reasonably fast, making bit_buf_type be long and setting BIT_BUF_SIZE
 * appropriately should be a win. Unfortunately we can't define the size
 * with something like #define BIT_BUF_SIZE (sizeof(bit_buf_type)*8)
 * because not all machines measure sizeof in 8-bit bytes.
 */

```

```

typedef struct { /* Bitreading state saved across MCUs */
    bit_buf_type get_buffer; /* Current bit-extraction buffer */
    int bits_left; /* # of unused bits in it */
} bitread_perm_state;

typedef struct { /* Bitreading working state within an MCU */
    /* Current data source location */
    /* We need a copy, rather than munging the original, in case of suspension */
    const JOCTET * next_input_byte; /* => next byte to read from source */
    size_t bytes_in_buffer; /* # of bytes remaining in source buffer */
    /* Bit input buffer --- note these values are kept in register variables,
     * not in this struct, inside the inner loops.
     */
    bit_buf_type get_buffer; /* current bit-extraction buffer */
    int bits_left; /* # of unused bits in it */
    /* Pointer needed by jpeg_fill_bit_buffer. */
    j_decompress_ptr cinfo; /* back link to decompress master record */
} bitread_working_state;

/* Macros to declare and load/save bitread local variables. */
#define BITREAD_STATE_VARS \
    register bit_buf_type get_buffer; \
    register int bits_left; \
    bitread_working_state br_state

#define BITREAD_LOAD_STATE(cinfo,permstate) \
    br_state.cinfo = cinfo; \
    br_state.next_input_byte = cinfo->src->next_input_byte; \
    br_state.bytes_in_buffer = cinfo->src->bytes_in_buffer; \
    get_buffer = permstate.get_buffer; \
    bits_left = permstate.bits_left;

#define BITREAD_SAVE_STATE(cinfo,permstate) \
    cinfo->src->next_input_byte = br_state.next_input_byte; \
    cinfo->src->bytes_in_buffer = br_state.bytes_in_buffer; \
    permstate.get_buffer = get_buffer; \
    permstate.bits_left = bits_left

/*
 * These macros provide the in-line portion of bit fetching.
 * Use CHECK_BIT_BUFFER to ensure there are N bits in get_buffer
 * before using GET_BITS, PEEK_BITS, or DROP_BITS.
 * The variables get_buffer and bits_left are assumed to be locals,
 * but the state struct might not be (jpeg_huff_decode needs this).
 * CHECK_BIT_BUFFER(state,n,action);
 *   Ensure there are N bits in get_buffer; if suspend, take action.
 *   val = GET_BITS(n);
 *   Fetch next N bits.
 *   val = PEEK_BITS(n);
 *   Fetch next N bits without removing them from the buffer.
 * DROP_BITS(n);
 *   Discard next N bits.
 * The value N should be a simple variable, not an expression, because it
 * is evaluated multiple times.
 */

#define CHECK_BIT_BUFFER(state,nbits,action) \
    { if (bits_left < (nbits)) { \
        if (! jpeg_fill_bit_buffer(&(amp;state),get_buffer,bits_left,nbits)) \
            { action; } \
        get_buffer = (state).get_buffer; bits_left = (state).bits_left; } }

#define GET_BITS(nbits) \
    (((int) (get_buffer >> (bits_left -= (nbits)))) & ((1<<(nbits))-1))

#define PEEK_BITS(nbits) \
    (((int) (get_buffer >> (bits_left - (nbits)))) & ((1<<(nbits))-1))

#define DROP_BITS(nbits) \
    (bits_left -= (nbits))

/* Load up the bit buffer to a depth of at least nbits */
EXTERN(boolean) jpeg_fill_bit_buffer
JPP((bitread_working_state * state, register bit_buf_type get_buffer,
    register int bits_left, int nbits));

/*
 * Code for extracting next Huffman-coded symbol from input bit stream.

```

\* Again, this is time-critical and we make the main paths be macros.  
 \* We use a lookahead table to process codes of up to HUFF\_LOOKAHEAD bits  
 \* without looping. Usually, more than 95% of the Huffman codes will be 8  
 \* or fewer bits long. The few overlength codes are handled with a loop,  
 \* which need not be inline code.

\* Notes about the HUFF\_DECODE macro:

- \* 1. Near the end of the data segment, we may fail to get enough bits  
 for a lookahead. In that case, we do it the hard way.
- \* 2. If the lookahead table contains no entry, the next code must be  
 more than HUFF\_LOOKAHEAD bits long.
- \* 3. jpeg\_huff\_decode returns -1 if forced to suspend.

\*/

```
#define HUFF_DECODE(result,state,htbl,failaction,slowlabel) \
{ register int nb, look; \
  if (bits_left < HUFF_LOOKAHEAD) { \
    if (! jpeg_fill_bit_buffer(&state,get_buffer,bits_left, 0)) {failaction;} \
    get_buffer = state.get_buffer; bits_left = state.bits_left; \
    if (bits_left < HUFF_LOOKAHEAD) { \
      nb = 1; goto slowlabel; \
    } \
  } \
  look = PEEK_BITS(HUFF_LOOKAHEAD); \
  if ((nb = htbl->look_nbits[look]) != 0) { \
    DROP_BITS(nb); \
    result = htbl->look_sym[look]; \
  } else { \
    nb = HUFF_LOOKAHEAD+1; \
slowlabel: \
    if ((result=jpeg_huff_decode(&state,get_buffer,bits_left,htbl,nb)) < 0) \
    { failaction; } \
    get_buffer = state.get_buffer; bits_left = state.bits_left; \
  } \
}
```

/\* Out-of-line case for Huffman code fetching \*/

EXTERN(int) jpeg\_huff\_decode

```
JPP((bitread_working_state * state, register bit_buf_type get_buffer,  

    register int bits_left, d_derived_tbl * htbl, int min_bits));
```

```

/*
 * jerror.h
 *
 * Copyright (C) 1994-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file defines the error and message codes for the JPEG library.
 * Edit this file to add new codes, or to translate the message strings to
 * some other language.
 * A set of error-reporting macros are defined too. Some applications using
 * the JPEG library may wish to include this file to get the error codes
 * and/or the macros.
 */

/*
 * To define the enum list of message codes, include this file without
 * defining macro JMESSAGE. To create a message string table, include it
 * again with a suitable JMESSAGE definition (see jerror.c for an example).
 */
#ifndef JMESSAGE
#define JMESSAGE
#endif

#ifndef JERROR_H
#define JERROR_H
/* First time through, define the enum list */
#define JMAKE_ENUM_LIST
#else
/* Repeated inclusions of this file are no-ops unless JMESSAGE is defined */
#define JMESSAGE(code,string)
#endif /* JERROR_H */
#endif /* JMESSAGE */

#ifdef JMAKE_ENUM_LIST

typedef enum {
#define JMESSAGE(code,string) code,
JMESSAGE(JMSG_NOMESSAGE, "Bogus message code %d") /* Must be first entry! */

/* For maintenance convenience, list is alphabetical by message code name */
JMESSAGE(JERR_ARITH_NOTIMPL,
  "Sorry, there are legal restrictions on arithmetic coding")
JMESSAGE(JERR_BAD_ALIGN_TYPE, "ALIGN_TYPE is wrong, please fix")
JMESSAGE(JERR_BAD_ALLOC_CHUNK, "MAX_ALLOC_CHUNK is wrong, please fix")
JMESSAGE(JERR_BAD_BUFFER_MODE, "Bogus buffer control mode")
JMESSAGE(JERR_BAD_COMPONENT_ID, "Invalid component ID %d in SOS")
JMESSAGE(JERR_BAD_DCT_COEF, "DCT coefficient out of range")
JMESSAGE(JERR_BAD_DCTSIZE, "IDCT output block size %d not supported")
JMESSAGE(JERR_BAD_HUFF_TABLE, "Bogus Huffman table definition")
JMESSAGE(JERR_BAD_IN_COLORSPACE, "Bogus input colorspace")
JMESSAGE(JERR_BAD_J_COLORSPACE, "Bogus JPEG colorspace")
JMESSAGE(JERR_BAD_LENGTH, "Bogus marker length")
JMESSAGE(JERR_BAD_LIB_VERSION,
  "Wrong JPEG library version: library is %d, caller expects %d")
JMESSAGE(JERR_BAD_MCU_SIZE, "Sampling factors too large for interleaved scan")
JMESSAGE(JERR_BAD_POOL_ID, "Invalid memory pool code %d")
JMESSAGE(JERR_BAD_PRECISION, "Unsupported JPEG data precision %d")
JMESSAGE(JERR_BAD_PROGRESSION,
  "Invalid progressive parameters Ss=%d Se=%d Ah=%d Al=%d")
JMESSAGE(JERR_BAD_PROG_SCRIPT,
  "Invalid progressive parameters at scan script entry %d")
JMESSAGE(JERR_BAD_SAMPLING, "Bogus sampling factors")
JMESSAGE(JERR_BAD_SCAN_SCRIPT, "Invalid scan script at entry %d")
JMESSAGE(JERR_BAD_STATE, "Improper call to JPEG library in state %d")
JMESSAGE(JERR_BAD_STRUCT_SIZE,
  "JPEG parameter struct mismatch: library thinks size is %u, caller expects %u")
JMESSAGE(JERR_BAD_VIRTUAL_ACCESS, "Bogus virtual array access")
JMESSAGE(JERR_BUFFER_SIZE, "Buffer passed to JPEG library is too small")
JMESSAGE(JERR_CANT_SUSPEND, "Suspension not allowed here")
JMESSAGE(JERR_CCIR601_NOTIMPL, "CCIR601 sampling not implemented yet")
JMESSAGE(JERR_COMPONENT_COUNT, "Too many color components: %d, max %d")
JMESSAGE(JERR_CONVERSION_NOTIMPL, "Unsupported color conversion request")
JMESSAGE(JERR_DAC_INDEX, "Bogus DAC index %d")
JMESSAGE(JERR_DAC_VALUE, "Bogus DAC value 0x%x")
JMESSAGE(JERR_DHT_INDEX, "Bogus DHT index %d")
JMESSAGE(JERR_DQT_INDEX, "Bogus DQT index %d")
JMESSAGE(JERR_EMPTY_IMAGE, "Empty JPEG image (DNL not supported)")
JMESSAGE(JERR_EMS_READ, "Read from EMS failed")
JMESSAGE(JERR_EMS_WRITE, "Write to EMS failed")

```



```

JMESSAGE(JERR_EOI_EXPECTED, "Don't expect more than one scan")
JMESSAGE(JERR_FILE_READ, "Input file read error")
JMESSAGE(JERR_FILE_WRITE, "Output file write error --- out of disk space?")
JMESSAGE(JERR_FRACT_SAMPLE_NOTIMPL, "Fractional sampling not implemented yet")
JMESSAGE(JERR_HUFF_CLEN_OVERFLOW, "Huffman code size table overflow")
JMESSAGE(JERR_HUFF_MISSING_CODE, "Missing Huffman code table entry")
JMESSAGE(JERR_IMAGE_TOO_BIG, "Maximum supported image dimension is %u pixels")
JMESSAGE(JERR_INPUT_EMPTY, "Empty input file")
JMESSAGE(JERR_INPUT_EOF, "Premature end of input file")
JMESSAGE(JERR_MISMATCHED_QUANT_TABLE,
    "Cannot transcode due to multiple use of quantization table %d")
JMESSAGE(JERR_MISSING_DATA, "Scan script does not transmit all data")
JMESSAGE(JERR_MODE_CHANGE, "Invalid color quantization mode change")
JMESSAGE(JERR_NOTIMPL, "Not implemented yet")
JMESSAGE(JERR_NOT_COMPILED, "Requested feature was omitted at compile time")
JMESSAGE(JERR_NO_BACKING_STORE, "Backing store not supported")
JMESSAGE(JERR_NO_HUFF_TABLE, "Huffman table 0x%02x was not defined")
JMESSAGE(JERR_NO_IMAGE, "JPEG datastream contains no image")
JMESSAGE(JERR_NO_QUANT_TABLE, "Quantization table 0x%02x was not defined")
JMESSAGE(JERR_NO_SOI, "Not a JPEG file: starts with 0x%02x 0x%02x")
JMESSAGE(JERR_OUT_OF_MEMORY, "Insufficient memory (case %d)")
JMESSAGE(JERR_QUANT_COMPONENTS,
    "Cannot quantize more than %d color components")
JMESSAGE(JERR_QUANT_FEW_COLORS, "Cannot quantize to fewer than %d colors")
JMESSAGE(JERR_QUANT_MANY_COLORS, "Cannot quantize to more than %d colors")
JMESSAGE(JERR_SOF_DUPLICATE, "Invalid JPEG file structure: two SOF markers")
JMESSAGE(JERR_SOF_NO_SOS, "Invalid JPEG file structure: missing SOS marker")
JMESSAGE(JERR_SOF_UNSUPPORTED, "Unsupported JPEG process: SOF type 0x%02x")
JMESSAGE(JERR_SOI_DUPLICATE, "Invalid JPEG file structure: two SOI markers")
JMESSAGE(JERR_SOS_NO_SOF, "Invalid JPEG file structure: SOS before SOF")
JMESSAGE(JERR_TFILE_CREATE, "Failed to create temporary file %s")
JMESSAGE(JERR_TFILE_READ, "Read failed on temporary file")
JMESSAGE(JERR_TFILE_SEEK, "Seek failed on temporary file")
JMESSAGE(JERR_TFILE_WRITE,
    "Write failed on temporary file --- out of disk space?")
JMESSAGE(JERR_TOO_LITTLE_DATA, "Application transferred too few scanlines")
JMESSAGE(JERR_UNKNOWN_MARKER, "Unsupported marker type 0x%02x")
JMESSAGE(JERR_VIRTUAL_BUG, "Virtual array controller messed up")
JMESSAGE(JERR_WIDTH_OVERFLOW, "Image too wide for this implementation")
JMESSAGE(JERR_XMS_READ, "Read from XMS failed")
JMESSAGE(JERR_XMS_WRITE, "Write to XMS failed")
JMESSAGE(JMSG_COPYRIGHT, JCOPYRIGHT)
JMESSAGE(JMSG_VERSION, JVERSION)
JMESSAGE(JTRC_16BIT_TABLES,
    "Caution: quantization tables are too coarse for baseline JPEG")
JMESSAGE(JTRC_ADOBE,
    "Adobe APP14 marker: version %d, flags 0x%04x 0x%04x, transform %d")
JMESSAGE(JTRC_APP0, "Unknown APP0 marker (not JFIF), length %u")
JMESSAGE(JTRC_APP14, "Unknown APP14 marker (not Adobe), length %u")
JMESSAGE(JTRC_DAC, "Define Arithmetic Table 0x%02x: 0x%02x")
JMESSAGE(JTRC_DHT, "Define Huffman Table 0x%02x")
JMESSAGE(JTRC_DQT, "Define Quantization Table %d precision %d")
JMESSAGE(JTRC_DRI, "Define Restart Interval %u")
JMESSAGE(JTRC_EMS_CLOSE, "Freed EMS handle %u")
JMESSAGE(JTRC_EMS_OPEN, "Obtained EMS handle %u")
JMESSAGE(JTRC_EOI, "End Of Image")
JMESSAGE(JTRC_HUFFBITS, "      %3d %3d %3d %3d %3d %3d %3d %3d")
JMESSAGE(JTRC_JFIF, "JFIF APP0 marker: version %d.%02d, density %dx%d %d")
JMESSAGE(JTRC_JFIF_BADTHUMBNAILSIZE,
    "Warning: thumbnail image size does not match data length %u")
JMESSAGE(JTRC_JFIF_EXTENSION,
    "JFIF extension marker: type 0x%02x, length %u")
JMESSAGE(JTRC_JFIF_THUMBNAIL, "      with %d x %d thumbnail image")
JMESSAGE(JTRC_MISC_MARKER, "Miscellaneous marker 0x%02x, length %u")
JMESSAGE(JTRC_PARMLESS_MARKER, "Unexpected marker 0x%02x")
JMESSAGE(JTRC_QUANTVALS, "      %4u %4u %4u %4u %4u %4u %4u %4u")
JMESSAGE(JTRC_QUANT_3_NCOLORS, "Quantizing to %d = %d*%d*%d colors")
JMESSAGE(JTRC_QUANT_NCOLORS, "Quantizing to %d colors")
JMESSAGE(JTRC_QUANT_SELECTED, "Selected %d colors for quantization")
JMESSAGE(JTRC_RECOVERY_ACTION, "At marker 0x%02x, recovery action %d")
JMESSAGE(JTRC_RST, "RST%d")
JMESSAGE(JTRC_SMOOTH_NOTIMPL,
    "Smoothing not supported with nonstandard sampling ratios")
JMESSAGE(JTRC_SOF, "Start Of Frame 0x%02x: width=%u, height=%u, components=%d")
JMESSAGE(JTRC_SOF_COMPONENT, "      Component %d: %dhx%dv q=%d")
JMESSAGE(JTRC_SOI, "Start of Image")
JMESSAGE(JTRC_SOS, "Start Of Scan: %d components")
JMESSAGE(JTRC_SOS_COMPONENT, "      Component %d: dc=%d ac=%d")
JMESSAGE(JTRC_SOS_PARAMS, "      Ss=%d, Se=%d, Ah=%d, Al=%d")
JMESSAGE(JTRC_TFILE_CLOSE, "Closed temporary file %s")

```

```

JMESSAGE(JTRC_TFILE_OPEN, "Opening temporary file %s")
JMESSAGE(JTRC_THUMB_JPEG,
    "JFIF extension marker: JPEG-compressed thumbnail image, length %u")
JMESSAGE(JTRC_THUMB_PALETTE,
    "JFIF extension marker: palette thumbnail image, length %u")
JMESSAGE(JTRC_THUMB_RGB,
    "JFIF extension marker: RGB thumbnail image, length %u")
JMESSAGE(JTRC_UNKNOWN_IDS,
    "Unrecognized component IDs %d %d %d, assuming YCbCr")
JMESSAGE(JTRC_XMS_CLOSE, "Freed XMS handle %u")
JMESSAGE(JTRC_XMS_OPEN, "Obtained XMS handle %u")
JMESSAGE(JWRN_ADOBE_XFORM, "Unknown Adobe color transform code %d")
JMESSAGE(JWRN_BOGUS_PROGRESSION,
    "Inconsistent progression sequence for component %d coefficient %d")
JMESSAGE(JWRN_EXTRANEEOUS_DATA,
    "Corrupt JPEG data: %u extraneous bytes before marker 0x%02x")
JMESSAGE(JWRN_HIT_MARKER, "Corrupt JPEG data: premature end of data segment")
JMESSAGE(JWRN_HUFF_BAD_CODE, "Corrupt JPEG data: bad Huffman code")
JMESSAGE(JWRN_JFIF_MAJOR, "Warning: unknown JFIF revision number %d.%02d")
JMESSAGE(JWRN_JPEG_EOF, "Premature end of JPEG file")
JMESSAGE(JWRN_MUST_RESYNC,
    "Corrupt JPEG data: found marker 0x%02x instead of RST%d")
JMESSAGE(JWRN_NOT_SEQUENTIAL, "Invalid SOS parameters for sequential JPEG")
JMESSAGE(JWRN_TOO_MUCH_DATA, "Application transferred too many scanlines")

```

```

#ifdef JMAKE_ENUM_LIST

```

```

    JMSG_LASTMSGCODE
} J_MESSAGE_CODE;

```

```

#undef JMAKE_ENUM_LIST

```

```

#endif /* JMAKE_ENUM_LIST */

```

```

/* Zap JMESSAGE macro so that future re-inclusions do nothing by default */

```

```

#undef JMESSAGE

```

```

#ifndef JERROR_H
#define JERROR_H

```

```

/* Macros to simplify using the error and trace message stuff */
/* The first parameter is either type of cinfo pointer */

```

```

/* Fatal errors (print message and exit) */

```

```

#define ERREXIT(cinfo,code) \
    ((cinfo)->err->msg_code = (code), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT1(cinfo,code,p1) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT2(cinfo,code,p1,p2) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT3(cinfo,code,p1,p2,p3) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (cinfo)->err->msg_parm.i[2] = (p3), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXIT4(cinfo,code,p1,p2,p3,p4) \
    ((cinfo)->err->msg_code = (code), \
    (cinfo)->err->msg_parm.i[0] = (p1), \
    (cinfo)->err->msg_parm.i[1] = (p2), \
    (cinfo)->err->msg_parm.i[2] = (p3), \
    (cinfo)->err->msg_parm.i[3] = (p4), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define ERREXITS(cinfo,code,str) \
    ((cinfo)->err->msg_code = (code), \
    strncpy((cinfo)->err->msg_parm.s, (str), JMSG_STR_PARM_MAX), \
    (*(cinfo)->err->error_exit) ((j_common_ptr) (cinfo)))

```

```

#define MAKESTMT(stuff)    do { stuff } while (0)

```

```

/* Nonfatal errors (we can keep going, but the data is probably corrupt) */

```

```

#define WARNMS(cinfo,code) \
    ((cinfo)->err->msg_code = (code), \
    (*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), -1))

```

```

#define WARNMS1(cinfo,code,p1) \
((cinfo)->err->msg_code = (code), \
(cinfo)->err->msg_parm.i[0] = (p1), \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), -1))
#define WARNMS2(cinfo,code,p1,p2) \
((cinfo)->err->msg_code = (code), \
(cinfo)->err->msg_parm.i[0] = (p1), \
(cinfo)->err->msg_parm.i[1] = (p2), \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), -1))

/* Informational/debugging messages */
#define TRACEMS(cinfo,lv1,code) \
((cinfo)->err->msg_code = (code), \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#define TRACEMS1(cinfo,lv1,code,p1) \
((cinfo)->err->msg_code = (code), \
(cinfo)->err->msg_parm.i[0] = (p1), \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#define TRACEMS2(cinfo,lv1,code,p1,p2) \
((cinfo)->err->msg_code = (code), \
(cinfo)->err->msg_parm.i[0] = (p1), \
(cinfo)->err->msg_parm.i[1] = (p2), \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#define TRACEMS3(cinfo,lv1,code,p1,p2,p3) \
MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
_mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); \
(cinfo)->err->msg_code = (code); \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMS4(cinfo,lv1,code,p1,p2,p3,p4) \
MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
_mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); _mp[3] = (p4); \
(cinfo)->err->msg_code = (code); \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMS5(cinfo,lv1,code,p1,p2,p3,p4,p5) \
MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
_mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); _mp[3] = (p4); \
_mp[4] = (p5); \
(cinfo)->err->msg_code = (code); \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMS8(cinfo,lv1,code,p1,p2,p3,p4,p5,p6,p7,p8) \
MAKESTMT(int * _mp = (cinfo)->err->msg_parm.i; \
_mp[0] = (p1); _mp[1] = (p2); _mp[2] = (p3); _mp[3] = (p4); \
_mp[4] = (p5); _mp[5] = (p6); _mp[6] = (p7); _mp[7] = (p8); \
(cinfo)->err->msg_code = (code); \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)); )
#define TRACEMSS(cinfo,lv1,code,str) \
((cinfo)->err->msg_code = (code), \
strncpy((cinfo)->err->msg_parm.s, (str), JMSG_STR_PARM_MAX), \
(*(cinfo)->err->emit_message) ((j_common_ptr) (cinfo), (lv1)))
#endif /* JERROR_H */

```

```

/*
 * jinclude.h
 *
 * Copyright (C) 1991-1994, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file exists to provide a single place to fix any problems with
 * including the wrong system include files. (Common problems are taken
 * care of by the standard jconfig symbols, but on really weird systems
 * you may have to edit this file.)
 *
 * NOTE: this file is NOT intended to be included by applications using the
 * JPEG library. Most applications need only include jpeglib.h.
 */

/* Include auto-config file to find out which system include files we need. */

#include "jconfig.h"      /* auto configuration options */
#define JCONFIG_INCLUDED  /* so that jpeglib.h doesn't do it again */

/*
 * We need the NULL macro and size_t typedef.
 * On an ANSI-conforming system it is sufficient to include <stddef.h>.
 * Otherwise, we get them from <stdlib.h> or <stdio.h>; we may have to
 * pull in <sys/types.h> as well.
 * Note that the core JPEG library does not require <stdio.h>;
 * only the default error handler and data source/destination modules do.
 * But we must pull it in because of the references to FILE in jpeglib.h.
 * You can remove those references if you want to compile without <stdio.h>.
 */
#ifdef HAVE_STDDEF_H
#include <stddef.h>
#endif
#ifdef HAVE_STDLIB_H
#include <stdlib.h>
#endif
#ifdef NEED_SYS_TYPES_H
#include <sys/types.h>
#endif
#include <stdio.h>

/*
 * We need memory copying and zeroing functions, plus strncpy().
 * ANSI and System V implementations declare these in <string.h>.
 * BSD doesn't have the mem() functions, but it does have bcopy()/bzero().
 * Some systems may declare memset and memcpy in <memory.h>.
 *
 * NOTE: we assume the size parameters to these functions are of type size_t.
 * Change the casts in these macros if not!
 */

#ifdef NEED_BSD_STRINGS

#include <strings.h>
#define MEMZERO(target,size)      bzero((void *) (target), (size_t) (size))
#define MEMCOPY(dest,src,size)    bcopy((const void *) (src), (void *) (dest), (size_t) (size))

#else /* not BSD, assume ANSI/SysV string lib */

#include <string.h>
#define MEMZERO(target,size)      memset((void *) (target), 0, (size_t) (size))
#define MEMCOPY(dest,src,size)    memcpy((void *) (dest), (const void *) (src), (size_t) (size))

#endif

/*
 * In ANSI C, and indeed any rational implementation, size_t is also the
 * type returned by sizeof(). However, it seems there are some irrational
 * implementations out there, in which sizeof() returns an int even though
 * size_t is defined as long or unsigned long. To ensure consistent results
 * we always use this SIZEOF() macro in place of using sizeof() directly.
 */
#define SIZEOF(object) ((size_t) sizeof(object))

```

```

/*
 * The modules that use fread() and fwrite() always invoke them through
 * these macros. On some systems you may need to twiddle the argument casts.
 * CAUTION: argument order is different from underlying functions!
 */

```

```
#define JFREAD(file,buf,sizeofbuf) \
    ((size_t) fread((void *) (buf), (size_t) 1, (size_t) (sizeofbuf), (file)))
#define JFWRITE(file,buf,sizeofbuf) \
    ((size_t) fwrite((const void *) (buf), (size_t) 1, (size_t) (sizeofbuf), (file)))
```

[illegible]

```

/*
 * jmemsys.h
 *
 * Copyright (C) 1992-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This include file defines the interface between the system-independent
 * and system-dependent portions of the JPEG memory manager.  No other
 * modules need include it.  (The system-independent portion is jmemmgr.c;
 * there are several different versions of the system-dependent portion.)
 *
 * This file works as-is for the system-dependent memory managers supplied
 * in the IJG distribution.  You may need to modify it if you write a
 * custom memory manager.  If system-dependent changes are needed in
 * this file, the best method is to #ifdef them based on a configuration
 * symbol supplied in jconfig.h, as we have done with USE_MSDOS_MEMMGR
 * and USE_MAC_MEMMGR.
 */

/* Short forms of external names for systems with brain-damaged linkers. */
#ifdef NEED_SHORT_EXTERNAL_NAMES
#define jpeg_get_small      jGetSmall
#define jpeg_free_small    jFreeSmall
#define jpeg_get_large     jGetLarge
#define jpeg_free_large    jFreeLarge
#define jpeg_mem_available jMemAvail
#define jpeg_open_backing_store jOpenBackStore
#define jpeg_mem_init      jMemInit
#define jpeg_mem_term      jMemTerm
#endif /* NEED_SHORT_EXTERNAL_NAMES */

/*
 * These two functions are used to allocate and release small chunks of
 * memory.  (Typically the total amount requested through jpeg_get_small is
 * no more than 20K or so; this will be requested in chunks of a few K each.)
 * Behavior should be the same as for the standard library functions malloc
 * and free; in particular, jpeg_get_small must return NULL on failure.
 * On most systems, these ARE malloc and free.  jpeg_free_small is passed the
 * size of the object being freed, just in case it's needed.
 * On an 80x86 machine using small-data memory model, these manage near heap.
 */
EXTERN(void *) jpeg_get_small JPP((j_common_ptr cinfo, size_t sizeofobject));
EXTERN(void) jpeg_free_small JPP((j_common_ptr cinfo, void * object,
                                  size_t sizeofobject));

/*
 * These two functions are used to allocate and release large chunks of
 * memory (up to the total free space designated by jpeg_mem_available).
 * The interface is the same as above, except that on an 80x86 machine,
 * far pointers are used.  On most other machines these are identical to
 * the jpeg_get/free_small routines; but we keep them separate anyway,
 * in case a different allocation strategy is desirable for large chunks.
 */
EXTERN(void *) jpeg_get_large JPP((j_common_ptr cinfo,
                                   size_t sizeofobject));
EXTERN(void) jpeg_free_large JPP((j_common_ptr cinfo, void * object,
                                  size_t sizeofobject));

/*
 * The macro MAX_ALLOC_CHUNK designates the maximum number of bytes that may
 * be requested in a single call to jpeg_get_large (and jpeg_get_small for that
 * matter, but that case should never come into play).  This macro is needed
 * to model the 64Kb-segment-size limit of far addressing on 80x86 machines.
 * On those machines, we expect that jconfig.h will provide a proper value.
 * On machines with 32-bit flat address spaces, any large constant may be used.
 *
 * NB: jmemmgr.c expects that MAX_ALLOC_CHUNK will be representable as type
 * size_t and will be a multiple of sizeof(aligned_type).
 */
#ifdef MAX_ALLOC_CHUNK /* may be overridden in jconfig.h */
#define MAX_ALLOC_CHUNK 1000000000L
#endif
#endif

```

```

/*
 * This routine computes the amount of space still available for allocation by
 * jpeg_get_large. If more space than this is needed, backing store will be
 * used. NOTE: any memory already allocated must not be counted.
 *
 * There is a minimum space requirement, corresponding to the minimum
 * feasible buffer sizes; jmemmgr.c will request that much space even if
 * jpeg_mem_available returns zero. The maximum space needed, enough to hold
 * all working storage in memory, is also passed in case it is useful.
 * Finally, the total space already allocated is passed. If no better
 * method is available, cinfo->mem->max_memory_to_use - already_allocated
 * is often a suitable calculation.
 *
 * It is OK for jpeg_mem_available to underestimate the space available
 * (that'll just lead to more backing-store access than is really necessary).
 * However, an overestimate will lead to failure. Hence it's wise to subtract
 * a slop factor from the true available space. 5% should be enough.
 *
 * On machines with lots of virtual memory, any large constant may be returned.
 * Conversely, zero may be returned to always use the minimum amount of memory.
 */

```

```

EXTERN(long) jpeg_mem_available JPP((j_common_ptr cinfo,
                                     long min_bytes_needed,
                                     long max_bytes_needed,
                                     long already_allocated));

```

```

/*
 * This structure holds whatever state is needed to access a single
 * backing-store object. The read/write/close method pointers are called
 * by jmemmgr.c to manipulate the backing-store object; all other fields
 * are private to the system-dependent backing store routines.
 */
#define TEMP_NAME_LENGTH 64 /* max length of a temporary file's name */

#ifdef USE_MSDOS_MEMMGR /* DOS-specific junk */
typedef unsigned short XMSH; /* type of extended-memory handles */
typedef unsigned short EMSH; /* type of expanded-memory handles */

typedef union {
    short file_handle; /* DOS file handle if it's a temp file */
    XMSH xms_handle; /* handle if it's a chunk of XMS */
    EMSH ems_handle; /* handle if it's a chunk of EMS */
} handle_union;

#endif /* USE_MSDOS_MEMMGR */

#ifdef USE_MAC_MEMMGR /* Mac-specific junk */
#include <Files.h>
#endif /* USE_MAC_MEMMGR */

```

```

typedef struct backing_store_struct * backing_store_ptr;

typedef struct backing_store_struct {
    /* Methods for reading/writing/closing this backing-store object */
    JMETHOD(void, read_backing_store, (j_common_ptr cinfo,
                                       backing_store_ptr info,
                                       void * buffer_address,
                                       long file_offset, long byte_count));
    JMETHOD(void, write_backing_store, (j_common_ptr cinfo,
                                       backing_store_ptr info,
                                       void * buffer_address,
                                       long file_offset, long byte_count));
    JMETHOD(void, close_backing_store, (j_common_ptr cinfo,
                                       backing_store_ptr info));

    /* Private fields for system-dependent backing-store management */
#ifdef USE_MSDOS_MEMMGR
    /* For the MS-DOS manager (jmemdos.c), we need: */
    handle_union handle; /* reference to backing-store storage object */
    char temp_name[TEMP_NAME_LENGTH]; /* name if it's a file */
#else
#ifdef USE_MAC_MEMMGR
    /* For the Mac manager (jmemmac.c), we need: */
    short temp_file; /* file reference number to temp file */

```

```

    FSSpec tempSpec; /* the FSSpec for the temp file */
    char temp_name[TEMP_NAME_LENGTH]; /* name if it's a file */
#else
    /* For a typical implementation with temp files, we need: */
    FILE * temp_file; /* stdio reference to temp file */
    char temp_name[TEMP_NAME_LENGTH]; /* name of temp file */
#endif
#endif
) backing_store_info;

/*
 * Initial opening of a backing-store object. This must fill in the
 * read/write/close pointers in the object. The read/write routines
 * may take an error exit if the specified maximum file size is exceeded.
 * (If jpeg_mem_available always returns a large value, this routine can
 * just take an error exit.)
 */

EXTERN(void) jpeg_open_backing_store JPP((j_common_ptr cinfo,
                                         backing_store_ptr info,
                                         long total_bytes_needed));

/*
 * These routines take care of any system-dependent initialization and
 * cleanup required. jpeg_mem_init will be called before anything is
 * allocated (and, therefore, nothing in cinfo is of use except the error
 * manager pointer). It should return a suitable default value for
 * max_memory_to_use; this may subsequently be overridden by the surrounding
 * application. (Note that max_memory_to_use is only important if
 * jpeg_mem_available chooses to consult it ... no one else will.)
 * jpeg_mem_term may assume that all requested memory has been freed and that
 * all opened backing-store objects have been closed.
 */
EXTERN(long) jpeg_mem_init JPP((j_common_ptr cinfo));
EXTERN(void) jpeg_mem_term JPP((j_common_ptr cinfo));

```



```

/*
 * jmorecfg.h
 *
 * Copyright (C) 1991-1997, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains additional configuration options that customize the
 * JPEG software for special applications or support machine-dependent
 * optimizations. Most users will not need to touch this file.
 */

/*
 * Define BITS_IN_JSAMPLE as either
 *   8   for 8-bit sample values (the usual setting)
 *  12   for 12-bit sample values
 * Only 8 and 12 are legal data precisions for lossy JPEG according to the
 * JPEG standard, and the IJG code does not support anything else!
 * We do not support run-time selection of data precision, sorry.
 */

#define BITS_IN_JSAMPLE 8 /* use 8 or 12 */

/*
 * Maximum number of components (color channels) allowed in JPEG image.
 * To meet the letter of the JPEG spec, set this to 255. However, darn
 * few applications need more than 4 channels (maybe 5 for CMYK + alpha
 * mask). We recommend 10 as a reasonable compromise; use 4 if you are
 * really short on memory. (Each allowed component costs a hundred or so
 * bytes of storage, whether actually used in an image or not.)
 */
#define MAX_COMPONENTS 10 /* maximum number of image components */

/*
 * Basic data types.
 * You may need to change these if you have a machine with unusual data
 * type sizes; for example, "char" not 8 bits, "short" not 16 bits,
 * or "long" not 32 bits. We don't care whether "int" is 16 or 32 bits,
 * but it had better be at least 16.
 */

/* Representation of a single sample (pixel element value).
 * We frequently allocate large arrays of these, so it's important to keep
 * them small. But if you have memory to burn and access to char or short
 * arrays is very slow on your hardware, you might want to change these.
 */

#if BITS_IN_JSAMPLE == 8
/* JSAMPLE should be the smallest type that will hold the values 0..255.
 * You can use a signed char by having GETJSAMPLE mask it with 0xFF.
 */

#ifdef HAVE_UNSIGNED_CHAR
typedef unsigned char JSAMPLE;
#define GETJSAMPLE(value) ((int) (value))

#else /* not HAVE_UNSIGNED_CHAR */
typedef char JSAMPLE;
#define GETJSAMPLE(value) ((int) (value) & 0xFF)

#endif /* HAVE_UNSIGNED_CHAR */

#else /* BITS_IN_JSAMPLE != 8 */
#define GETJSAMPLE(value) ((int) (value))

#endif /* BITS_IN_JSAMPLE == 8 */

#define MAXJSAMPLE 255
#define CENTERJSAMPLE 128

#if BITS_IN_JSAMPLE == 8
/* JSAMPLE should be the smallest type that will hold the values 0..4095.
 */

```

```

* On nearly all machines "short" will do nicely.
*/

typedef short JSAMPLE;
#define GETJSAMPLE(value) ((int) (value))

#define MAXJSAMPLE 4095
#define CENTERJSAMPLE 2048

#endif /* BITS_IN_JSAMPLE == 12 */

/* Representation of a DCT frequency coefficient.
* This should be a signed value of at least 16 bits; "short" is usually OK.
* Again, we allocate large arrays of these, but you can change to int
* if you have memory to burn and "short" is really slow.
*/

typedef short JCOEF;

/* Compressed datastreams are represented as arrays of JOCTET.
* These must be EXACTLY 8 bits wide, at least once they are written to
* external storage. Note that when using the stdio data source/destination
* managers, this is also the data type passed to fread/fwrite.
*/

#ifdef HAVE_UNSIGNED_CHAR
typedef unsigned char JOCTET;
#define GETJOCTET(value) (value)
#else /* not HAVE_UNSIGNED_CHAR */
typedef char JOCTET;
#ifdef CHAR_IS_UNSIGNED
#define GETJOCTET(value) (value)
#else
#define GETJOCTET(value) ((value) & 0xFF)
#endif /* CHAR_IS_UNSIGNED */
#endif /* HAVE_UNSIGNED_CHAR */

/*
* These typedefs are used for various table entries and so forth.
* They must be at least as wide as specified; but making them too big
* won't cost a huge amount of memory, so we don't provide special
* extraction code like we did for JSAMPLE. (In other words, these
* typedefs live at a different point on the speed/space tradeoff curve.)
*/

/*
* UINT8 must hold at least the values 0..255. */
#ifdef HAVE_UNSIGNED_CHAR
typedef unsigned char UINT8;
#else /* not HAVE_UNSIGNED_CHAR */
#ifdef CHAR_IS_UNSIGNED
typedef char UINT8;
#else /* not CHAR_IS_UNSIGNED */
typedef short UINT8;
#endif /* CHAR_IS_UNSIGNED */
#endif /* HAVE_UNSIGNED_CHAR */

/*
* UINT16 must hold at least the values 0..65535. */
#ifdef HAVE_UNSIGNED_SHORT
typedef unsigned short UINT16;
#else /* not HAVE_UNSIGNED_SHORT */
typedef unsigned int UINT16;
#endif /* HAVE_UNSIGNED_SHORT */

/*
* INT16 must hold at least the values -32768..32767. */
#ifdef XMD_H /* X11/xmd.h correctly defines INT16 */
typedef short INT16;
#else
typedef int INT16;
#endif

/*
* INT32 must hold at least signed 32-bit values. */
#ifdef XMD_H /* X11/xmd.h correctly defines INT32 */

```

```

typedef int INT32;
#endif

/* Datatype used for image dimensions. The JPEG standard only supports
 * images up to 64K*64K due to 16-bit fields in SOF markers. Therefore
 * "unsigned int" is sufficient on all machines. However, if you need to
 * handle larger images and you don't mind deviating from the spec, you
 * can change this datatype.
 */

typedef unsigned int JDIMENSION;

#define JPEG_MAX_DIMENSION 65500L /* a tad under 64K to prevent overflows */

/* These macros are used in all function definitions and extern declarations.
 * You could modify them if you need to change function linkage conventions;
 * in particular, you'll need to do that to make the library a Windows DLL.
 * Another application is to make all functions global for use with debuggers
 * or code profilers that require it.
 */

/* a function called through method pointers: */
#define METHODDEF(type) static type
/* a function used only in its module: */
#define LOCAL(type) static type
/* a function referenced thru EXTERNS: */
#define GLOBAL(type) type
/* a reference to a GLOBAL function: */
#define EXTERN(type) extern type

/* This macro is used to declare a "method", that is, a function pointer.
 * We want to supply prototype parameters if the compiler can cope.
 * Note that the arglist parameter must be parenthesized!
 * Again, you can customize this if you need special linkage keywords.
 */
#ifdef HAVE_PROTOTYPES
#define JMETHOD(type,methodname,arglist) type (*methodname) arglist
#else
#define JMETHOD(type,methodname,arglist) type (*methodname) ()
#endif

/* Here is the pseudo-keyword for declaring pointers that must be "far"
 * on 80x86 machines. Most of the specialized coding for 80x86 is handled
 * by just saying "FAR *" where such a pointer is needed. In a few places
 * explicit coding is needed; see uses of the NEED_FAR_POINTERS symbol.
 */
#ifdef INTIAZ : commented this out.
#ifdef NEED_FAR_POINTERS
#define FAR far
#else
#define FAR far
#endif
#endif

/*
 * On a few systems, type boolean and/or its values FALSE, TRUE may appear
 * in standard header files. Or you may have conflicts with application-
 * specific header files that you want to include together with these files.
 * Defining HAVE_BOOLEAN before including jpeglib.h should make it work.
 */
#ifdef HAVE_BOOLEAN
typedef int boolean;
#endif
#endif
#define FALSE 0 /* in case these macros already exist */
#define TRUE 1 /* values of boolean */

/*
 * The remaining options affect code selection within the JPEG library,

```

```

* but they don't need to be visible to most applications using the library.
* To minimize application namespace pollution, the symbols won't
* be defined unless JPEG_INTERNALS or JPEG_INTERNAL_OPTIONS has been defined.
*/

```

```

#ifdef JPEG_INTERNALS
#define JPEG_INTERNAL_OPTIONS
#endif

```

```

#ifdef JPEG_INTERNAL_OPTIONS

```

```

/*
 * These defines indicate whether to include various optional functions.
 * Undefined some of these symbols will produce a smaller but less capable
 * library. Note that you can leave certain source files out of the
 * compilation/linking process if you've #undef'd the corresponding symbols.
 * (You may HAVE to do that if your compiler doesn't like null source files.)
 */

```

```

/* Arithmetic coding is unsupported for legal reasons. Complaints to IBM. */

```

```

/* Capability options common to encoder and decoder: */

```

```

#define DCT_ISLOW_SUPPORTED /* slow but accurate integer algorithm */
#define DCT_IFAST_SUPPORTED /* faster, less accurate integer method */
#define DCT_FLOAT_SUPPORTED /* floating-point: accurate, fast on fast HW */

```

```

/* Encoder capability options: */

```

```

#undef C_ARITH_CODING_SUPPORTED /* Arithmetic coding back end? */
#define C_MULTISCAN_FILES_SUPPORTED /* Multiple-scan JPEG files? */
#define C_PROGRESSIVE_SUPPORTED /* Progressive JPEG? (Requires MULTISCAN) */
#define ENTROPY_OPT_SUPPORTED /* Optimization of entropy coding parms? */
/* Note: if you selected 12-bit data precision, it is dangerous to turn off
 * ENTROPY_OPT_SUPPORTED. The standard Huffman tables are only good for 8-bit
 * precision, so jchuff.c normally uses entropy optimization to compute
 * usable tables for higher precision. If you don't want to do optimization,
 * you'll have to supply different default Huffman tables.
 * The exact same statements apply for progressive JPEG: the default tables
 * don't work for progressive mode. (This may get fixed, however.)
 */
#define INPUT_SMOOTHING_SUPPORTED /* Input image smoothing option? */

```

```

/* Decoder capability options: */

```

```

#undef D_ARITH_CODING_SUPPORTED /* Arithmetic coding back end? */
#define D_MULTISCAN_FILES_SUPPORTED /* Multiple-scan JPEG files? */
#define D_PROGRESSIVE_SUPPORTED /* Progressive JPEG? (Requires MULTISCAN) */
#define SAVE_MARKERS_SUPPORTED /* jpeg_save_markers() needed? */
#define BLOCK_SMOOTHING_SUPPORTED /* Block smoothing? (Progressive only) */
#define IDCT_SCALING_SUPPORTED /* Output rescaling via IDCT? */
#undef UPSAMPLE_SCALING_SUPPORTED /* Output rescaling at upsample stage? */
#define UPSAMPLE_MERGING_SUPPORTED /* Fast path for sloppy upsampling? */
#define QUANT_1PASS_SUPPORTED /* 1-pass color quantization? */
#define QUANT_2PASS_SUPPORTED /* 2-pass color quantization? */

```

```

/* more capability options later, no doubt */

```

```

/*
 * Ordering of RGB data in scanlines passed to or from the application.
 * If your application wants to deal with data in the order B,G,R, just
 * change these macros. You can also deal with formats such as R,G,B,X
 * (one extra byte per pixel) by changing RGB_PIXELSIZE. Note that changing
 * the offsets will also change the order in which colormap data is organized.
 * RESTRICTIONS:
 * 1. The sample applications cjpeg,djpeg do NOT support modified RGB formats.
 * 2. These macros only affect RGB<=>YCbCr color conversion, so they are not
 * useful if you are using JPEG color spaces other than YCbCr or grayscale.
 * 3. The color quantizer modules will not behave desirably if RGB_PIXELSIZE
 * is not 3 (they don't understand about dummy color components!). So you
 * can't use color quantization if you change that value.
 */

```

```

#define RGB_RED 0 /* Offset of Red in an RGB scanline element */
#define RGB_GREEN 1 /* Offset of Green */
#define RGB_BLUE 2 /* Offset of Blue */
#define RGB_PIXELSIZE 3 /* JSAMPLES per RGB scanline element */

```

```

/* Definitions for speed-related optimizations. */

/* If your compiler supports inline functions, define INLINE
 * as the inline keyword; otherwise define it as empty.
 */

#ifndef INLINE
#ifdef __GNUC__ /* for instance, GNU C knows about inline */
#define INLINE __inline__
#endif
#ifndef INLINE
#define INLINE /* default is to define it as empty */
#endif
#endif

/* On some machines (notably 68000 series) "int" is 32 bits, but multiplying
 * two 16-bit shorts is faster than multiplying two ints. Define MULTIPLIER
 * as short on such a machine. MULTIPLIER must be at least 16 bits wide.
 */

#ifndef MULTIPLIER
#define MULTIPLIER int /* type for fastest integer multiply */
#endif

/* FAST_FLOAT should be either float or double, whichever is done faster
 * by your compiler. (Note that this type is only used in the floating point
 * DCT routines, so it only matters if you've defined DCT_FLOAT_SUPPORTED.)
 * Typically, float is faster in ANSI C compilers, while double is faster in
 * pre-ANSI compilers (because they insist on converting to double anyway).
 * The code below therefore chooses float if we have ANSI-style prototypes.
 */
#ifndef FAST_FLOAT
#ifdef HAVE_PROTOTYPES
#define FAST_FLOAT float
#else
#define FAST_FLOAT double
#endif
#endif

#endif /* JPEG_INTERNAL_OPTIONS */

```

```

    for(i=1;i<4;i++) r_Leye[i]=fabs(r_Leye[i]);
}
v1=sqrt(r_Leye[1]*r_Leye[1]+r_Leye[2]*r_Leye[2]+r_Leye[3]*r_Leye[3]);
for (i=1;i<4;i++) { r_Leye[i]/=v1 ; LL[i]=1/r_Leye[i]; }
if(how=='Y') LLL=LL[1]/(r_ALscale[1]*LL[1]);
v3=hypot(r_Leye[1],r_Leye[2]) ; v2=-r_Leye[2]*r_Leye[3]/v3 ; v1=-r_Leye[1]*r_Leye[3]/v3 ;
u1=r_Leye[2]/v3 ; u2=-r_Leye[1]/v3 ;
a=Na[2]*u2-Na[1]*u1 ; b=Na[3]*v3-Na[2]*v2-Na[1]*v1;
m1=(int)(__min(r_LE/r_DIM,r_HI*a/(b*r_DIM))) ; m2=(int)(m1*b/a) ; du=__min(a/m1,b/m2) ;
a=r_DIM/du ; un1=u1*Na[1]*a ; un2=u2*Na[2]*a ;
vn1=v1*Na[1]*a ; vn2=v2*Na[2]*a ; vn3=v3*Na[3]*a ;

NX[0]=0; NX[1]=(int)(r_HI+vn2);
NX[2]=0; NX[3]=(int)(r_HI+vn2-vn3);
NX[4]=(int)(-un1); NX[5]=(int)(r_HI+vn1+vn2-vn3);
NX[6]=(int)(m1*r_DIM-1); NX[7]=(int)(r_HI-v1*(NX[6]-un2)/u1-vn3);
NX[8]=NX[6]; NX[9]=(int)(r_HI-v1*(NX[8]-un2)/u1);
NX[10]=(int)(un2); NX[11]=(int)(r_HI);
NX[12]=NX[0]; NX[13]=NX[1];
a=__max((double)(r_HI)/(r_MAXY-25),(double)(r_LE)/(r_MAXX-225)) ;
for(i=1;i<=13;i+=2)
{
    nx[i]=(int)(NX[i]/a+20);
    nx[i-1]=(int)(NX[i-1]/a+220);
}
//!!setfillstyle(EMPTY_FILL,0) ; bar(214,17,r_MAXX,r_MAXY);
if ((m1<=3)|| (m2<=3)|| (du<0.001))
{
    sprintf(r_error,"Rendered image is too small");
    return false;
}
//!!setlinestyle(SOLID_LINE,0,NORM_WIDTH); rectangle (216,19,224+r_LE/a,24+r_HI/a);
//!!drawpoly(7,nx);
b=220-un1/a; c=20+(r_HI+vn1+vn2)/a;
//!!line(b,c,nx[0],nx[1]); line(b,c,nx[4],nx[5]);line(b,c,nx[8],nx[9]);
//!!bar(b,c,b+8,c+8); bar(nx[0],nx[1],nx[0]+16,nx[1]+8);
//!!bar(nx[4],nx[5],nx[4]+16,nx[5]+8); bar(nx[8]-8,nx[9],nx[8]+8,nx[9]+8);
//!!outtextxy(b,c,"O");
//!!if(r_OZ>0) outtextxy(nx[4],nx[5],"z"); else outtextxy(nx[4],nx[5],"-z");
//!!switch(r_OX) {
//!!case 1: outtextxy(nx[0],nx[1],"x"); outtextxy(nx[8]-8,nx[9],"y"); break;
//!!case -1: outtextxy(nx[0],nx[1],"-x"); outtextxy(nx[8]-8,nx[9],"-y"); break;
//!!case 2: outtextxy(nx[0],nx[1],"y"); outtextxy(nx[8]-8,nx[9],"-x"); break;
//!!case -2: outtextxy(nx[0],nx[1],"-y"); outtextxy(nx[8]-8,nx[9],"x"); break;
//!!}
b=220+un2/a; c=20+(r_HI-vn3)/a;
//!!line (b,c,220+un2/a,20+r_HI/a); line (b,c,220,20+(r_HI+vn2-vn3)/a);
//!!line (b,c,nx[8],nx[7]); setlinestyle(SOLID_LINE,0,THICK_WIDTH);
CU1=u1; CU2=u2; CV1=v1; CV2=v2; CV3=v3;
AP[1]=du; AP[2]=un1; AP[3]=un2;
CF1=vn1; CF2=vn2; CF3=vn3;
r_DIM1=m1; r_DIM2=m2;
return true;
}

/*****
*
* Find cube sign with respect to the current surface level.
* Returns: 0-equal signsof the cube vertices
* 1-different signs
*
*****/
bool RayTracer::CUBESIGN(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8)
{
    if(
        ((a1>r_LEV)&&(a2>r_LEV)&&(a3>r_LEV)&&(a4>r_LEV)&&
         (a5>r_LEV)&&(a6>r_LEV)&&(a7>r_LEV)&&(a8>r_LEV)) ||
        ((a1<r_LEV)&&(a2<r_LEV)&&(a3<r_LEV)&&(a4<r_LEV)&&
         (a5<r_LEV)&&(a6<r_LEV)&&(a7<r_LEV)&&(a8<r_LEV))) return false;
}

```

```

else return true;
}

```

```

/*****
 *
 *   MOVING RAY, JUMPING FROM ONE EDGE TO ANOTHER,
 *   AND SEEKING FOR ROOT, IF IT EXISTS.
 *   RETURNS: DEPTH>=0 - BODY'S FOUND.
 *             -1 - BODY WASN'T FOUND
 *
 *****/

```

```

double RayTracer::JUMP(double t)
{
    int      m,k,k1,i,i1,fin,h,sp[4];
    long     pfh,qfh,pd,qd;
    long     dt=0;
    double   d,f,fh,a,b;

    k1=0;
    /* LET'S JUMP : */
    jump: i=TOP[1] ; k=TOP[2] ; m=TOP[3] ;
    if (CUBESIGN(F(i,k,m),F(i,k+1,m),F(i,k,m+1),F(i,k+1,m+1),F(i+1,k,m),
        F(i+1,k+1,m),F(i+1,k,m+1),F(i+1,k+1,m+1))=0)
    {
        /* JUMP'S STEP: */
        h=_min(_min(P[1],P[2]),P[3]) ;
        for(i1=1;i1<4;i1++) {
            if(h==P[i1]) { if(P[i1]==-IL[i1]) NX[i1]--;
                else P[i1]=-IL[i1] ;
                if(NX[i1]<=0) return (-1);
                TOP[i1]=NX[i1]-1; IED[i1]=i1;
            }
            else { if(P[i1]==-IL[i1]) { if(NX[i1]<1) return (-1) ;
                NX[i1]--;
            }
            P[i1]-=h ; IED[i1]=0; TOP[i1]=NX[i1] ;
        }
        /* next i1 */
        k1=0 ;
        /* end of going trough cube with equal signs of tops */
    }
    else
    {
        if(k1!=0) { pd=pfh; qd=qfh; }
        else { EDGE() ; pd=PE; qd=QE; if(pd==0) return ( t+dt); }
        fin=0 ;

        /* JUMP'S STEP: */
        h=_min(_min(P[1],P[2]),P[3]) ;
        for(i1=1;i1<4;i1++) {
            sp[i1]=P[i1];
            if(h==P[i1]) { if(P[i1]==-IL[i1]) NX[i1]--;
                else P[i1]=-IL[i1] ;
                if(NX[i1]<=0) fin=1;
                TOP[i1]=NX[i1]-1; IED[i1]=i1;
            }

            else { if(P[i1]==-IL[i1]) { if(NX[i1]<1) return (-1);
                NX[i1]--;
            }
            P[i1]-=h ; IED[i1]=0 ; TOP[i1]=NX[i1] ; sp[i1]+=P[i1];
        }
        /* next i1 */
        EDGE(); pfh=PE; qfh=QE; /* pfh/qfh = PE/QE */
        if( ((pfh>0)&&(pd<0)) || ((pfh<0)&&(pd>0)) ) {
            f=FUNC(sp[1],sp[2],sp[3]); d=(double)(pd)/qd ; fh=(double)(pfh)/qfh ;
            if(f==fh) return ( t+dt+0.5*h*d/(d-f) );
            if(f==d) return ( t+dt+h*(f-0.5*fh)/(f-fh) );
            a=f/(fh-d); b=d-f; b+=b; b=fh/b;
            return ( t+dt+(a+b)*h*d/(fh-f) );
        }
    }
}

```

```

    }

if( pfh==0 ) { f=FUNC(sp[1],sp[2],sp[3]);
    if( ((f>=0)&&(pd<0))||((f<=0)&&(pd>0)) ) {
        d=(double)(pd)/qd; return ( t+dt+(0.5*h*d)/(d-2*f) );
    }
    else return ( t+(dt+h) );
}

k1=1 ; if(fin==1) return (-1) ;
} /* end of going trough the cube with different signs of tops */

dt+=h;
goto jump;
}

/*****
*
* MOVING RAY,JUMPING FROM ONE EDGE TO ANOTHER,
* AND SEEKING FOR ROOT, IF IT EXISTS.
* RETURNS:      +1 - BODY'S FOUND.
*              -1 - BODY WASN'T FOUND
*
*****/
int RayTracer::Q_JUMP()
{
    int m,k,k1,i,il,fin,h;
    long fh,d;

    k1=0;
    /* LET'S Q_JUMP : */
    jump: i=TOP[1] ; k=TOP[2] ; m=TOP[3] ;
    if(CUBESIGN(F(i,k,m),F(i,k+1,m),F(i,k,m+1),F(i,k+1,m+1),F(i+1,k,m),
        F(i+1,k+1,m),F(i+1,k,m+1),F(i+1,k+1,m+1))==0)
    {
        /* Q_JUMP'S STEP: */
        h=__min(__min(P[1],P[2]),P[3]) ;
        for(il=1;il<4;il++) {
            if(h==P[il]) { if(P[il]==-IL[il]) NX[il]--;
                else P[il]=-IL[il] ;
                if(NX[il]<=0) return (-1);
                TOP[il]=NX[il]-1; IED[il]=il;
            }

            else { if(P[il]==-IL[il]) { if(NX[il]<1) return (-1) ;
                NX[il]--;
            }
            P[il]-=h ; IED[il]=0; TOP[il]=NX[il] ;
        }
        /* next il */

        k1=0 ;
    } /* end of going trough cube with equal signs of tops */

else
{
    if(k1!=0) d=fh; else { EDGE() ; d=PE; }
    fin=0 ;

    /* Q_JUMP'S STEP: */
    h=__min(__min(P[1],P[2]),P[3]) ;
    for(il=1;il<4;il++) {
        if(h==P[il]) { if(P[il]==-IL[il]) NX[il]--;
            else P[il]=-IL[il] ;
            if(NX[il]<=0) fin=1;
            TOP[il]=NX[il]-1; IED[il]=il;
        }

        else { if(P[il]==-IL[il]) { if(NX[il]<1) return (-1);

```



```

        NX[i1]--;
    }
    P[i1]--=h ; IED[i1]=0 ; TOP[i1]=NX[i1] ;
    }
    } /* next i1 */
EDGE(); fh=PE;
    if(((d>=0)&&(fh<=0))||((d<=0)&&(fh>=0))) return(+1);
kl=1 ; if(fin==1) return (-1) ;
    } /* end of going through the cube with different signs of tops */

goto qjump;
}

/*..... */
/* CALCULATES FUNCTION'S VALUE ON THE EDGE : F = PE/QE
    IL12=IL[1]*IL[2] ; IL13=IL[1]*IL[3] ; IL23=IL[2]*IL[3] ; */
void RayTracer::EDGE()
{
    int i,j,k;
    long z,a,b;
    i=NX[1] ; j=NX[2] ; k=NX[3] ; z=F(i,j,k) ;
    switch (IED[1]+IED[2]-IED[3]) {
    case -1: PE=P[1]*(F(i+1,j,k)-z)-(z-r_LEV)*IL[1]; QE=-IL[1]; break; /* 0 2 3 */
    case -2: PE=P[2]*(F(i,j+1,k)-z)-(z-r_LEV)*IL[2]; QE=-IL[2]; break; /* 1 0 3 */
    case 3: PE=P[3]*(F(i,j,k+1)-z)-(z-r_LEV)*IL[3]; QE=-IL[3]; break; /* 1 2 0 */
    case -3:
        a=(z-F(i+1,j,k)-F(i,j+1,k)+F(i+1,j+1,k))*P[2]-(F(i+1,j,k)-z)*IL[2] ;
        b=(F(i,j+1,k)-z)*P[2]-(z-r_LEV)*IL[2];
        PE=a*P[1]-b*IL[1]; QE=IL12; break; /* 0 0 3 */
    case 2:
        a=(z-F(i+1,j,k)-F(i,j,k+1)+F(i+1,j,k+1))*P[3]-(F(i+1,j,k)-z)*IL[3] ;
        b=(F(i,j,k+1)-z)*P[3]-(z-r_LEV)*IL[3];
        PE=a*P[1]-b*IL[1]; QE=IL13; break; /* 0 2 0 */
    case 1:
        a=(z-F(i,j+1,k)-F(i,j,k+1)+F(i,j+1,k+1))*P[3]-(F(i,j+1,k)-z)*IL[3] ;
        b=(F(i,j,k+1)-z)*P[3]-(z-r_LEV)*IL[3];
        PE=a*P[2]-b*IL[2]; QE=IL23; break; /* 1 0 0 */
    default: PE=z-r_LEV; QE=1; break;
    }
    return;
}

/*..... */
/* CALCULATES FUNCTION VALUE IN THE POINT (x,y,z)
    RETURNS- FUNCTION VALUE */
double RayTracer::FUNC(long x, long y, long z)
{
    int i,j,k,f,f1;
    long a,b;
    i=NX[1] ; j=NX[2] ; k=NX[3] ; f=F(i,j,k) ; f1=F(i,j,k+1);
    a=x*((f+F(i+1,j+1,k)-F(i+1,j,k)-F(i,j+1,k))*y+(F(i+1,j,k)-f)*DL2 )
        +DL1*( (F(i,j+1,k)-f)*y+(f-r_LEV)*DL2 );
    b=x*((f1+F(i+1,j+1,k+1)-F(i+1,j,k+1)-F(i,j+1,k+1))*y+DL2*(F(i+1,j,k+1)
        -f1))+DL1*( (F(i,j+1,k+1)-f1)*y+(f1-r_LEV)*DL2 );
    return ( a*D1+(b-a)*(z*D2) );
}

/*..... */
/* INSTALL YOUR POINT ON THE CUBE EDGE (X,Y,Z- DISTANCES)
    RETURNS: IS[p][q] (IF POINT CAN'T BE INSTALLED,
        FILLS IS[p][q]=-1 )
    */
void RayTracer::INST(int p, int q, int u, int v)
{
    int i;
    double a,h,e[4];

```

```

IS[p][q]=-1;
u+=p; v+=q;
e[1]=CU1*u+CV1*v+CF1; e[2]=CU2*u+CV2*v+CF2; e[3]=CV3*v+CF3;
h=__max(__max(e[1],e[2]),e[3]);
for(i=1;i<4;i++) {
    if(h==e[i]) { NX[i]=r_N[i]; P[i]=-IL[i]; IED[i]=i; TOP[i]=r_N[i]-1; }
    else { a=AP[i]+e[i]-h; if(a<0.5) return;
        AA=(long)a; NX[i]=AA/(-IL[i]); P[i]=AA+(long)(NX[i])*IL[i];
        if(P[i]==0) { if(NX[i]==0) { P[i]=1; IED[i]=0; TOP[i]=0; }
            else { P[i]=-IL[i]; IED[i]=i; TOP[i]=NX[i]-1; }
        }
    }
    else { IED[i]=0; TOP[i]=NX[i]; }
}
} /* next i */
IS[p][q]=Q_JUMP (); return ;
}

/*..... */
/*CALCULATE INTENSITY:
    b

a <-r_DIM-> t          c

d

RETURN:
VAR=>r_COLb*BD , IF BODY EXISTS
0          IF BODY WAS NOT FOUND
*/
int RayTracer::INTENS(double a, double b, double c, double d, double t)
{
    double nor1,nor2;
    if (t<0) return 0;
    if(a>=0)
    {
        if(c>=0) nor1=c-a;
        else     nor1=2*(t-a);
    }
    else
    {
        if(c>=0) nor1=2*(c-t);
        else     nor1=0;
    }

    if(d>=0)
    {
        if(b>=0) nor2=b-d;
        else     nor2=2*(t-d);
    }
    else
    {
        if(b>=0) nor2=2*(b-t);
        else     nor2=0;
    }
    return ( (int)( B1+B2/sqrt(nor1*nor1+nor2*nor2+DOWN) ) );
}

/*..... */
/*ESTABLISH SURFACE PRESENCE FILLING IS[] [] ARRAY
*
*
*

```

```

*/
void RayTracer::CHESS(int uu, int vv, int i1, int i2, int i3, int i4,
                      int j1, int j2, int j3, int j4, int j5, int j6,
                      int j7, int j8)
{
    signed char p,q,p0;
    // Set IS to unidentified "-2"
    p0=IS[0][0];
    for(p=0;p<r_DIM;p++){ for(q=0;q<r_DIM;q++) IS[p][q]=-2; }
    IS[0][0]=p0;
    // Get corners
    INST(0,r_DIM1,uu,vv); INST(r_DIM1,0,uu,vv); INST(r_DIM1,r_DIM1,uu,vv);
    // Is DIM small ?
    if(r_DIM<=2)
    {
        GURO2 ('*',i1,i2,i3,i4,j1,j2,j3,j4,j5,j6,j7,j8,uu,vv) ;
        return ;
    }
    // DIM is >2; do the "chess" work
    p0=IS[0][0];
    if( p0==IS[0][r_DIM1] && p0==IS[r_DIM1][0] && p0==IS[r_DIM1][r_DIM1] )
    {
        for(p=0;p<r_DIM;p++){ for(q=0;q<r_DIM;q++) IS[p][q]=p0; }
        GURO2 ('*',i1,i2,i3,i4,j1,j2,j3,j4,j5,j6,j7,j8,uu,vv) ;
        return ;
    }
    q=0; p0=0;
    while (q<r_DIM)
    {
        for(p=p0;p<r_DIM;p+=2)
        {
            if(IS[p][q]!=-2) continue;
            else INST(p,q,uu,vv);
        }
        p0=1-p0; q++;
    }
    for(p=1;p<r_DIM1;p+=2)
    {
        if(IS[p-1][0]==IS[p][1] && IS[p-1][0]==IS[p+1][0]) IS[p][0]=IS[p-1][0];
        else INST(p,0,uu,vv);
    }
    for(q=1;q<r_DIM1;q+=2)
    {
        if(IS[0][q-1]==IS[1][q] && IS[0][q-1]==IS[0][q+1]) IS[0][q]=IS[0][q-1];
        else INST(0,q,uu,vv);
    }
    for(p=(r_DIM1%2)+1;p<r_DIM1;p+=2)
    {
        if(IS[p-1][r_DIM1]==IS[p][r_DIM1-1] && IS[p-1][r_DIM1]==IS[p+1][r_DIM1])
        { IS[p][r_DIM1]=IS[p-1][r_DIM1]; }
        else INST(p,r_DIM1,uu,vv);
    }
    for(q=(r_DIM1%2)+1;q<r_DIM1;q+=2)
    {
        if(IS[r_DIM1][q-1]==IS[r_DIM1-1][q] && IS[r_DIM1][q-1]==IS[r_DIM1][q+1])
        { IS[r_DIM1][q]=IS[r_DIM1][q-1]; }
        else INST(r_DIM1,q,uu,vv);
    }
    q=1; p0=1;
    while (q<r_DIM1)
    {
        for(p=p0+1;p<r_DIM1;p+=2)
        {
            if(IS[p][q]!=-2) continue;
            if( IS[p-1][q]==IS[p+1][q] && IS[p-1][q]==IS[p][q-1] &&
                IS[p-1][q]==IS[p][q+1]) IS[p][q]=IS[p-1][q];
            else INST(p,q,uu,vv);
        }
        p0=1-p0; q++;
    }
    GURO2 ('*',i1,i2,i3,i4,j1,j2,j3,j4,j5,j6,j7,j8,uu,vv) ; return ;
}

```

```

/*..... */
/* FILLS SQUARE WITH GURO2 INTENSITY:          J1<-r_DIM->J2
MASK:  '-' -LIGHT
      '*' -AS IN IS' CODE                      J8    I3    I4    J3

                        J7    I1    I2    J4
                        (ix,iy)

                        J6    J5

RETURNS PICTURE                                     */

```

```

void RayTracer::GURO2(char mask, long i1, long i2, long i3, long i4,
                      long j1, long j2, long j3, long j4, long j5,
                      long j6, long j7, long j8, int ix, int iy)
{
    int x,y,xx,yy,choix=0;
    long a12,a21,a20,a02,a11,a01,a10,a00,pas,pasy,in,debut,kpas,lpas;
    long kdebut,ldebut,lpasy;
    if(mask=='*') {
        if(i1>0) choix=1 ; if(i2>0) choix+=10 ;
        if(i3>0) choix+=100; if(i4>0) choix+=1000;
        switch(choix) {
            case 1:  i2=i1 ; i3=i1 ; i4=i1 ;
                    if(j1<=0) j1=i1; if(j2<=0) j2=i1; if(j3<=0) j3=i1;
                    if(j4<=0) j4=i1; if(j5<=0) j5=i1; if(j6<=0) j6=i1;
                    if(j7<=0) j7=i1; if(j8<=0) j8=i1;
                    break;
            case 10: i1=i2 ; i3=i2 ; i4=i2 ;
                    if(j1<=0) j1=i2; if(j2<=0) j2=i2; if(j3<=0) j3=i2;
                    if(j4<=0) j4=i2; if(j5<=0) j5=i2; if(j6<=0) j6=i2;
                    if(j7<=0) j7=i2; if(j8<=0) j8=i2;
                    break;
            case 11: i3=i1 ; i4=i2 ;
                    if(j1<=0) j1=i1; if(j2<=0) j2=i2; if(j3<=0) j3=i2;
                    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=i2; if(j6<=0) j6=i1;
                    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=i1;
                    break;
            case 100: i1=i3 ; i2=i3 ; i4=i3 ;
                    if(j1<=0) j1=i3; if(j2<=0) j2=i3; if(j3<=0) j3=i3;
                    if(j4<=0) j4=i3; if(j5<=0) j5=i3; if(j6<=0) j6=i3;
                    if(j7<=0) j7=i3; if(j8<=0) j8=i3;
                    break;
            case 101: i2=i1 ; i4=i3 ;
                    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=i3; if(j3<=0) j3=i3;
                    if(j4<=0) j4=i1; if(j5<=0) j5=i1; if(j6<=0) j6=__min(i1,i3);
                    if(j7<=0) j7=i1; if(j8<=0) j8=i3;
                    break;
            case 110: i1=(i2+i3)/2 ; i4=i1 ;
                    if(j1<=0) j1=i3; if(j2<=0) j2=i2; if(j3<=0) j3=i3;
                    if(j4<=0) j4=i2; if(j5<=0) j5=i2; if(j6<=0) j6=i3;
                    if(j7<=0) j7=i2; if(j8<=0) j8=i3;
                    break;
            case 111: i4=__min(i2,i3) ;
                    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=i2; if(j3<=0) j3=i3;
                    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=i2; if(j6<=0) j6=__min(i1,i3);
                    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=i3;
                    break;
            case 1000: i1=i4 ; i2=i4 ; i3=i4 ;
                    if(j1<=0) j1=i4; if(j2<=0) j2=i4; if(j3<=0) j3=i4;
                    if(j4<=0) j4=i4; if(j5<=0) j5=i4; if(j6<=0) j6=i4;
                    if(j7<=0) j7=i4; if(j8<=0) j8=i4;
                    break;
            case 1001: i2=(i1+i4)/2 ; i3=i2 ;
                    if(j1<=0) j1=i1; if(j2<=0) j2=i4; if(j3<=0) j3=i4;
                    if(j4<=0) j4=i1; if(j5<=0) j5=i4; if(j6<=0) j6=i1;
                    if(j7<=0) j7=i1; if(j8<=0) j8=i4;
                    break;
            case 1010: i1=i2 ; i3=i4 ;
                    if(j1<=0) j1=i4; if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=i4;
                    if(j4<=0) j4=i2; if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=i2;
                    if(j7<=0) j7=i2; if(j8<=0) j8=i4;
                    break;

```

```

case 1011: i3=__min(i1,i4);
    if(j1<=0) j1=i1; if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=__min(i3,i4);
    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=i1;
    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=i4;
    break;
case 1100: i1=i3 ; i2=i4 ;
    if(j1<=0) j1=i3; if(j2<=0) j2=i4; if(j3<=0) j3=__min(i3,i4);
    if(j4<=0) j4=i4; if(j5<=0) j5=i4; if(j6<=0) j6=i3;
    if(j7<=0) j7=i3; if(j8<=0) j8=__min(i3,i4);
    break;
case 1101: i2=__min(i1,i4) ;
    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=i4; if(j3<=0) j3=__min(i3,i4);
    if(j4<=0) j4=i1; if(j5<=0) j5=i4; if(j6<=0) j6=__min(i1,i3);
    if(j7<=0) j7=i1; if(j8<=0) j8=__min(i3,i4);
    break;
case 1110: i1=__min(i2,i3) ;
    if(j1<=0) j1=i3; if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=__min(i3,i4);
    if(j4<=0) j4=i2; if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=i3;
    if(j7<=0) j7=i2; if(j8<=0) j8=__min(i3,i4);
    break;
default : return;
} /* end of switch */

a00=4*i1*r_DIM3 ; a10=(-3*i1+5*i2-j7-j4)*r_DIM2 ; a01=(-3*i1+5*i3-j1-j6)*r_DIM2 ;
a02=(-i1-i3+j1+j6)*r_DIM ; a20=(-i1-i2+j4+j7)*r_DIM ;
a21=i1+i2-i3-i4+j3-j4-j7+j8 ; a12=i1-i2+i3-i4-j1+j2+j5-j6 ;
a11=(2*i1-4*i2-4*i3+6*i4+j1-j2-j3+j4-j5+j6+j7-j8)*r_DIM ;
pas=a02+a01 ; kpas=2*a12 ; lpas=a21+a12+a11 ;
debut=a00 ; kdebut=2*a20 ; ldebut=a10+a20 ;
lpasy=2*(a12+a02) ;

for (x=0,xx=ix;x<r_DIM;x++,xx++) {
    pasy=pas ; in=debut;
    for (y=0,yy=r_HI-iy;y<r_DIM;y++,yy--) {
        if (IS[x][y]>0)
        {
            int pix=in/BD;
            r_Screen->SetPixel(xx,yy,pix);
        }
        in+=pasy;
        pasy+=lpasy;
    }
    pas+=kpas*x+lpas;
    debut+=kdebut*x+ldebut;
}
return; } /* end of mask='*' */

else {
    if(j1<=0) j1=__min(i1,i3); if(j2<=0) j2=__min(i2,i4); if(j3<=0) j3=__min(i3,i4);
    if(j4<=0) j4=__min(i1,i2); if(j5<=0) j5=__min(i2,i4); if(j6<=0) j6=__min(i3,i1);
    if(j7<=0) j7=__min(i1,i2); if(j8<=0) j8=__min(i3,i4);
    a00=4*i1*r_DIM3 ; a10=(-3*i1+5*i2-j7-j4)*r_DIM2 ; a01=(-3*i1+5*i3-j1-j6)*r_DIM2 ;
    a02=(-i1-i3+j1+j6)*r_DIM ; a20=(-i1-i2+j4+j7)*r_DIM ;
    a21=i1+i2-i3-i4+j3-j4-j7+j8 ; a12=i1-i2+i3-i4-j1+j2+j5-j6 ;
    a11=(2*i1-4*i2-4*i3+6*i4+j1-j2-j3+j4-j5+j6+j7-j8)*r_DIM ;
    pas=a02+a01 ; kpas=2*a12 ; lpas=a21+a12+a11 ;
    debut=a00 ; kdebut=2*a20 ; ldebut=a10+a20 ;
    lpasy=2*(a12+a02) ;

    int imax=4*r_DIM3*__max(__max(i1,i2),__max(i3,i4))/BD;
    int imin=4*r_DIM3*__min(__min(i1,i2),__min(i3,i4))/BD;
    for (x=0,xx=ix;x<r_DIM;x++,xx++) {
        pasy=pas ; in=debut;
        for (y=0,yy=r_HI-iy;y<r_DIM;y++,yy--) {
            int pix=in/BD;
            if(pix>imax) pix=imax; else if(pix<imin) pix=imin;
            r_Screen->SetPixel(xx,yy,pix);
            in+=pasy;
            pasy+=lpasy;
        }
        pas+=kpas*x+lpas;
        debut+=kdebut*x+ldebut;
    }
}

return;
} /* end of case mask != '*' */

```

```

}

/*****
 *
 *   Set voxel scaling
 *
 *****/
bool RayTracer::SetVolumeScales(double sx, double sy, double sz)
{
    double smin=__min(sx,__min(sy,sz));
    double smax=__max(sx,__max(sy,sz));
    if(smin<=0)
    {
        sprintf(r_error, "Non-positive volume scale");
        return false;
    }
    const int maxscale=64;
    double coef=maxscale/smax;
    r_ALscale[1]=__max(1,(int)__min(maxscale,sx*coef+0.5));
    r_ALscale[2]=__max(1,(int)__min(maxscale,sy*coef+0.5));
    r_ALscale[3]=__max(1,(int)__min(maxscale,sz*coef+0.5));
    return true;
}

/*****
 *
 *   Set observer's eye position
 *
 *****/
void RayTracer::SetEyePosition(double x, double y, double z)
{
    r_Leye[1]=x;
    r_Leye[2]=y;
    r_Leye[3]=z;
}

/*****
 *
 *   Set observer's eye position, where
 *   deg_hor and deg_ver are angles in radians
 *
 *****/
void RayTracer::SetEyePositionOnSphere(double deg_hor, double deg_ver)
{
    double c=cos(deg_ver);
    SetEyePosition(c*cos(deg_hor), c*sin(deg_hor), sin(deg_ver));
}

/*****
 *
 *   Set tracing granularity
 *
 *****/
void RayTracer::SetGranularity(int gr)
{
    r_DIM=gr;
    if(r_DIM>16)        r_DIM=16;
    else if(r_DIM<2)    r_DIM=2;
}

/*****
 *
 *   Set isosurface level to be visualized
 *
 *****/
void RayTracer::SetSurfaceLevel(int lev)    {    r_LEV=lev;    }

/*****
 *
 *   Set max and min body color

```

```

*
*****
void RayTracer::SetBodyColor(int cmin, int cmax)
{
    cmin=__max(cmin,0);
    if(cmin>cmax)
    {
        int t=cmin; cmin=cmax; cmax=t;
    }
    r_COLb=cmin; // body minimum
    r_COLf=__max(1,cmax-cmin); // body range: r_COLf=(foreground-r_COLb)
}

/*****
*
*   Return VOXEL value
*
*****/

int RayTracer::F(int x, int y, int z)
{
    /*
    switch(r_RotationCode)
    {
    case 001: return F_XpYpZm(x,y,z);
    case 010: return F_XpYmZp(x,y,z);
    case 011: return F_XpYmZm(x,y,z);
    case 100: return F_XmYpZp(x,y,z);
    case 101: return F_XmYpZm(x,y,z);
    case 110: return F_XmYmZp(x,y,z);
    case 111: return F_XmYmZm(x,y,z);
    default: return r_Volume->GetVoxel(x,y,z); //return F_XpYpZp(x,y,z);
    }
    */
    return (this->*F_XYZ)(x,y,z);
}

/*****
*
*   Functions to obtain voxel value for
*   different volume orientation.
*   Array r_N[] must be already modified
*
*****/

int RayTracer::F_XpYpZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(x,y,z);
}
int RayTracer::F_XmYpZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(y,r_N[2]-x,z);
}
int RayTracer::F_XpYmZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-y,x,z);
}
int RayTracer::F_XmYmZp(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-x,r_N[2]-y,z);
}
int RayTracer::F_XpYpZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(y,x,r_N[3]-z);
}
int RayTracer::F_XmYpZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-x,y,r_N[3]-z);
}
int RayTracer::F_XpYmZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(x,r_N[2]-y,r_N[3]-z);
}
int RayTracer::F_XmYmZm(int x, int y, int z)
{
    return r_Volume->GetVoxel(r_N[1]-y,r_N[2]-x,r_N[3]-z);
}

```

$\bar{G}$ [illegible]



```

// Volume.cpp: implementation of the Volume class.
//
/////////////////////////////////////////////////////////////////

#include "Volume.h"

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

Volume::Volume()
{
    m_Fmax=-30000;  m_Fmin=30000;
}

Volume::~Volume()
{
}

/*****
 *
 *   Functions returning volume dimensions - OVERRIDE (virtual)
 *
 *****/
int Volume::GetXSize() { return 40; }
int Volume::GetYSize() { return 40; }
int Volume::GetZSize() { return 40; }

/*****
 *
 *   Function returning voxel - OVERRIDE (virtual)
 *
 *****/
int Volume::GetVoxel(int x, int y, int z)
{
    return ((x-20)*(x-20)+(y-20)*(y-20)+(z-20)*(z-20)-500)/3;
    //return (x+y+z);
}

/*****
 *
 *   Find min and max voxel values
 *
 *****/
void Volume::FindMinMax()
{
    if(m_Fmax>m_Fmin) return; // already found

    int i,j,k,vox;
    m_Fmax=m_Fmin=GetVoxel(0,0,0);
    for(i=0; i<GetXSize(); i++)
    {
        for(j=0; j<GetYSize(); j++)
        {
            for(k=0; k<GetZSize(); k++)
            {
                vox=GetVoxel(i,j,k);
                if(vox>m_Fmax) m_Fmax=vox;
                else if(vox<m_Fmin) m_Fmin=vox;
            }
        }
    }
    if(m_Fmax<=m_Fmin) m_Fmax=m_Fmin+1;
}

/*****
 *
 *   Return min and max voxel values
 *
 *****/
void Volume::GetMinMax(int &fmin, int &fmax)
{
    FindMinMax();
}

```

```
fmin=m_Fmin;    fmax=m_Fmax;
```

1

//

```
#define AFX_VOLUME_H__1FBBBE11_9D18_11D3_8140_000000000000__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

1

```
void GetMinMax(int& fmin, int& fmax);
virtual int GetVoxel(int x, int y, int z);
virtual int GetZSize();
virtual int GetYSize();
virtual int GetXSize();
Volume();
virtual ~Volume();
```

```
void FindMinMax();
int m_Fmin, m_Fmax;
```

```
#endif // !defined(AFX_VOLUME_H__1FBBBE11_9D18_11D3_8140_000000000000__INCLUDED_)
```

The following table shows the results of the regression analysis for the dependent variable *Perceived Stress*. The independent variables are *Age*, *Gender*, *Marital Status*, *Income*, *Education*, *Health Status*, *Work Status*, *Family Size*, *Religious Beliefs*, *Social Support*, *Life Events*, *Coping Strategies*, *Personality Traits*, *Resilience*, *Optimism*, *Pessimism*, *Neuroticism*, *Extraversion*, *Conscientiousness*, *Agreeableness*, *Openness*, *Stress Management*, *Problem Solving*, *Emotional Regulation*, *Self-Efficacy*, *Perceived Stress*, *Life Satisfaction*, *Life Events*, *Coping Strategies*, *Personality Traits*, *Resilience*, *Optimism*, *Pessimism*, *Neuroticism*, *Extraversion*, *Conscientiousness*, *Agreeableness*, *Openness*, *Stress Management*, *Problem Solving*, *Emotional Regulation*, *Self-Efficacy*, *Perceived Stress*, *Life Satisfaction*.

```

// pixels.h
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_PIXELS_H__INCLUDED_
#define AFX_PIXELS_H__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "..\dicom.hpp"
#include "..\cctypes.h" // Added by ClassView
/////////////////////////////////////////////////////////////////
//
// RawPixelDecoder class.
//
/////////////////////////////////////////////////////////////////
class RawPixelDecoder
{
public:
    bool        Load_Pixels(FILE* infile);
    bool        Load_Pixels(BYTE* data, UINT32 data_size);
    long        GetPixelMinimum() { return m_PixMin; };
    long        GetPixelMaximum() { return m_PixMax; };
    inline long GetBufferedPixelSample(UINT32 n);
    inline long GetBufferedPixel(UINT32 n);

    RawPixelDecoder(UINT32 npixels, int bits_alloc,
                    int bits_stored, int high_bit,
                    int samples_per_pixel, int endian=1);
    virtual ~RawPixelDecoder();

private:
    bool        m_ReleaseBufferMemory;
    BYTE*       m_pBuffer;
    int         m_SamplesPerPixel;
    int         m_PixShift;
    int         m_LitEndian;
    int         m_BytesPerPixel;
    long        m_PixMin, m_PixMax;
    UINT32      m_nSamples;
    UINT32      m_PixMask;
    UINT32      m_nBytes;
    UINT32      m_nPixels;

    void        Digest();
    void        ReleaseBuffer();
    void        Initialize(UINT32 npixels, int bits_alloc,
                          int bits_stored, int high_bit,
                          int samples_per_pixel, int endian=1);
    bool        SetPixelMask(int bits_allocated,
                          int bits_stored, int high_bit);
    bool        Valid();
};

/*****
 *
 *   Get sample and pixel #n in little endian
 *
 *****/
inline long RawPixelDecoder::GetBufferedPixelSample(UINT32 n)
{
    if(n>=m_nSamples) n=m_nSamples-1; // safe
    n *= m_BytesPerPixel; // map to array byte address
    long p=m_pBuffer[n];
    switch(m_BytesPerPixel)
    {
    case 2:
        p=p | ( ((UINT32)m_pBuffer[n+1])<<8 );
        break;
    case 3:
        p=p | ( ((UINT32)m_pBuffer[n+1])<<8 );
        p=p | ( ((UINT32)m_pBuffer[n+2])<<16 );
        break;
    case 4:
        p=p | ( ((UINT32)m_pBuffer[n+1])<<8 );
        p=p | ( ((UINT32)m_pBuffer[n+2])<<16 );
        p=p | ( ((UINT32)m_pBuffer[n+3])<<24 );
        break;
    }
}

```

```

    }
    return (p>>m_PixShift)&m_Mask;
};
inline long RawPixelDecoder::GetBufferedPixel(UINT32 n)
{
    if(m_SamplesPerPixel==1)
    {
        return GetBufferedPixelSample(n);
    }
    else
    {
        n *= m_SamplesPerPixel;
        long p=GetBufferedPixelSample(n);
        for(int i=1; i<m_SamplesPerPixel; i++)
        {
            n++;
            p += GetBufferedPixelSample(n);
        }
        p /= m_SamplesPerPixel;
        return p;
    }
}

```

```

/////////////////////////////////////////////////////////////////
//
// RawPixelEncoder class.
//
/////////////////////////////////////////////////////////////////

```

```

class RawPixelEncoder
{
public:
    void      TransferDataToVR(VR *vr);
    bool      SetSize(UINT32 size);
    UINT32    AddData(BYTE* buf, UINT32 buf_size,
                     UINT32 fliprawbytes=0);

```

```

    RawPixelEncoder();
    virtual ~RawPixelEncoder();

```

```

private:
    bool      m_ReleaseBufferMemory;
    BYTE*     m_pBuffer;
    UINT32    m_BufferPtr;
    UINT32    m_Size;

    void      ReleaseBuffer();

```

```

#endif // !defined(AFX_PIXELS_H__2A361EE3_CEAB_11D3_AF48_000000000000__INCLUDED_)

```

```

// pixels.cpp
//
/////////////////////////////////////////////////////////////////

#include "pixels.h"

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////
RawPixelDecoder::RawPixelDecoder(UINT32 npixels, int bits_alloc,
                                int bits_stored, int high_bit,
                                int samples_per_pixel, int endian/*=1*/)
{
    m_BytesPerPixel=0;
    m_LitEndian=1;
    m_nBytes=m_nPixels=m_nSamples=m_SamplesPerPixel=0;
    m_pBuffer=NULL; m_ReleaseBufferMemory=false;
    m_PixMask=m_PixShift=0;
    Initialize( npixels, bits_alloc, bits_stored, high_bit,
               samples_per_pixel, endian );
}

RawPixelDecoder::~RawPixelDecoder() { ReleaseBuffer(); }

/*****
 *
 *   Initialize buffer parameters
 *
 *****/
void RawPixelDecoder::Initialize(UINT32 npixels, int bits_alloc,
                                int bits_stored, int high_bit,
                                int samples_per_pixel, int endian/*=1*/)
{
    m_ReleaseBufferMemory=false;
    m_nPixels=npixels; // number of pixels to read; one pixel may have several samples
    m_BytesPerPixel=(bits_alloc+7)/8; // number of bytes per pixel
    m_SamplesPerPixel=samples_per_pixel; // samples per pixel
    m_nSamples=m_nPixels*m_SamplesPerPixel;
    m_LitEndian=endian; // endian type
    m_nBytes=m_nSamples*m_BytesPerPixel; // total number of bytes
    if(!SetPixelMask(bits_alloc, bits_stored, high_bit)) m_BytesPerPixel=0;
}

/*****
 *
 *   Validate current parameter values
 *
 *****/
bool RawPixelDecoder::Valid()
{
    if( m_BytesPerPixel<=0 || m_BytesPerPixel>4 ||
        m_SamplesPerPixel<=0 || m_SamplesPerPixel>4 ||
        m_nPixels<=4 || m_nSamples<=4 || m_nBytes<=4 )
    {
        return false;
    }
    else return true;
}

/*****
 *
 *   Compute pixel mask parameters
 *
 *****/
bool RawPixelDecoder::SetPixelMask(int bits_allocated, int bits_stored, int high_bit)
{
    if( bits_allocated<bits_stored || bits_allocated<high_bit ||
        bits_allocated<=0 || bits_stored<=0 ) return false;

    m_PixMask=1;
    for(int i=1; i<bits_stored; i++)
    {
        m_PixMask = (m_PixMask<<1)+1;
    }
    m_PixShift=high_bit-bits_stored+1;
    return true;
}

/*****
 *
 *   Load pixel data, in bytes, from the input file or buffer
 *
 *****/

```

```

*
*****/
bool RawPixelDecoder::Load_Pixels(FILE * infile)
{
    if(!Valid())    return false;
    ReleaseBuffer();
    try
    {
        m_pBuffer=new BYTE[m_nBytes];    // pixel buffer
        if(!m_pBuffer) return false; // out of memory
    }
    catch(...) {    return false;    }
    memset(m_pBuffer,0,m_nBytes);
    m_ReleaseBufferMemory=true;
    if(fread(m_pBuffer,1,m_nBytes, infile) < m_nBytes) return false;
    Digest();
    return true;
}

bool RawPixelDecoder::Load_Pixels(BYTE *data, UINT32 data_size)
{
    if(!Valid())    return false;
    ReleaseBuffer();
    if(m_nBytes>data_size) return false;    // incomplete data
    m_ReleaseBufferMemory=false;
    m_pBuffer=data;
    Digest();
    return true;
}

}

/*****
*
* Process loaded pixel bytes. Return pixel range
*
*****/
void RawPixelDecoder::Digest()
{
    UINT32 i;
    // If not little endian - go swap bytes
    if(!m_LitEndian)
    {
        ::SwitchEndian(m_pBuffer, m_nBytes, m_BytesPerPixel);
    }

    // Find min and max pixel SAMPLE values
    long p;
    m_PixMin=GetBufferedPixelSample(0);
    m_PixMax=m_PixMin+1;
    for(i=1; i<m_nSamples; i++)
    {
        p=GetBufferedPixelSample(i);
        if(p>m_PixMax) m_PixMax=p;
        else if(p<m_PixMin) m_PixMin=p;
    }
}

/*****
*
* Release allocated buffer memory
*
*****/
void RawPixelDecoder::ReleaseBuffer()
{
    if(m_ReleaseBufferMemory)
    {
        if(m_pBuffer)
        {
            delete [] m_pBuffer;
            m_pBuffer=NULL;
        }
        m_ReleaseBufferMemory=false;
    }
}

```

```

int nXpos = 0;
int nYpos1 = nTop + 1;
int nYpos2 = nBottom - 2;

for (int i = 0; i < nNumBands; i++)
{
    nXpos = nAdjust + (i * IND_BAND_WIDTH);

    pDC->SelectObject(&m_penColorDarker);
    pDC->MoveTo(nXpos + 1, nTop);
    pDC->LineTo(nXpos + nHeight, nBottom);

    pDC->SelectObject(&m_penColorDark);
    pDC->MoveTo(nXpos + 2, nTop);
    pDC->LineTo(nXpos + nHeight + 1, nBottom);
    pDC->MoveTo(nXpos + 10, nTop);
    pDC->LineTo(nXpos + nHeight + 9, nBottom);

    pDC->SelectObject(&m_penColor);
    pDC->MoveTo(nXpos + 3, nTop);
    pDC->LineTo(nXpos + nHeight + 2, nBottom);
    pDC->MoveTo(nXpos + 9, nTop);
    pDC->LineTo(nXpos + nHeight + 8, nBottom);

    pDC->SelectObject(&m_penColorLight);
    pDC->MoveTo(nXpos + 4, nTop);
    pDC->LineTo(nXpos + nHeight + 3, nBottom);
    pDC->MoveTo(nXpos + 8, nTop);
    pDC->LineTo(nXpos + nHeight + 7, nBottom);

    pDC->SelectObject(&m_penColorLighter);
    pDC->MoveTo(nXpos + 5, nTop);
    pDC->LineTo(nXpos + nHeight + 4, nBottom);
    pDC->MoveTo(nXpos + 7, nTop);
    pDC->LineTo(nXpos + nHeight + 6, nBottom);
} // for the number of bands
} // if indeterminate
else
{
    int nRight = rect.right;

    pDC->MoveTo(nLeft + 2, nBottom - 4);
    pDC->LineTo(nRight - 2, nBottom - 4);
    pDC->MoveTo(nLeft + 2, nTop + 2);
    pDC->LineTo(nRight - 2, nTop + 2);
    pDC->SetPixel(nLeft + 1, nBottom - 3, m_crColorLight);
    pDC->SetPixel(nLeft + 1, nTop + 1, m_crColorLight);

    pDC->SelectObject(&m_penColorLighter);
    pDC->MoveTo(nLeft + 2, nBottom - 5);
    pDC->LineTo(nRight - 3, nBottom - 5);
    pDC->LineTo(nRight - 3, nTop + 3);
    pDC->LineTo(nLeft + 1, nTop + 3);
    pDC->SetPixel(nLeft + 1, nBottom - 4, m_crColorLighter);
    pDC->SetPixel(nLeft + 1, nTop + 2, m_crColorLighter);

    pDC->SelectObject(&m_penColor);
    pDC->MoveTo(nLeft, nBottom - 1);
    pDC->LineTo(nLeft, nTop);
    pDC->LineTo(nLeft + 2, nTop);
    pDC->SetPixel(nLeft + 1, nBottom - 2, m_crColor);
    pDC->MoveTo(nLeft + 2, nBottom - 3);
    pDC->LineTo(nRight - 2, nBottom - 3);
    pDC->MoveTo(nLeft + 2, nTop + 1);
    pDC->LineTo(nRight - 1, nTop + 1);

    pDC->SelectObject(&m_penColorDark);
    pDC->MoveTo(nLeft + 2, nBottom - 2);
    pDC->LineTo(nRight - 2, nBottom - 2);
    pDC->LineTo(nRight - 2, nTop + 1);
    pDC->MoveTo(nLeft + 2, nTop);
    pDC->LineTo(nRight, nTop);
    pDC->SetPixel(nLeft + 1, nBottom - 1, m_crColorDark);

```



```

pDC->SelectObject(&m_penColorDarker);
pDC->MoveTo(nLeft + 1, nBottom - 1);
pDC->LineTo(nRight - 1, nBottom - 1);
pDC->LineTo(nRight - 1, nTop);

```

```

pDC->SelectObject(&m_penShadow);
pDC->MoveTo(nRight, nTop);
pDC->LineTo(nRight, nBottom);

```

```

pDC->SelectObject(&m_penLiteShadow);
pDC->MoveTo(nRight + 1, nTop);
pDC->LineTo(nRight + 1, nBottom);

```

```

} // if not indeterminate

```

```

pDC->SelectObject(pOldPen);
} // DrawHorizontalBar

```

```

//-----

```

```

//
void CMacProgressCtrl::DrawVerticalBar(CDC *pDC, const CRect rect)
//
// Return Value:      None.
//
// Parameters       :   pDC - Specifies the device context object.
//                    rect - Specifies the rectangle of the progress bar.
//
// Remarks          :   Draws a vertical progress bar.
//
{

```

```

    int nHeight = rect.Height();
    if (!nHeight)
        return;

```

```

    int nLeft = rect.left;
    int nTop = rect.top;
    int nRight = rect.right;
    int nBottom = rect.bottom;

```

```

    CPen *pOldPen = pDC->SelectObject(&m_penColor);

```

```

    if (m_bIndeterminate)
    {
        int nNumBands = (nHeight / IND_BAND_WIDTH) + 2;
        int nHeight = rect.Width() + 1;

        int nAdjust = nBottom - m_nIndOffset;
        int nXpos1 = nLeft;
        int nXpos2 = nRight + 1;
        int nYpos = nTop + 1;

```

```

        for (int i = 0; i < nNumBands; i++)
        {
            nYpos = nAdjust - (i * IND_BAND_WIDTH);

```

```

            pDC->SelectObject(&m_penColorDarker);
            pDC->MoveTo(nXpos1, nYpos);
            pDC->LineTo(nXpos2, nYpos + nHeight);

```

```

            pDC->SelectObject(&m_penColorDark);
            pDC->MoveTo(nXpos1, nYpos + 1);
            pDC->LineTo(nXpos2, nYpos + nHeight + 1);
            pDC->MoveTo(nXpos1, nYpos + 9);
            pDC->LineTo(nXpos2, nYpos + nHeight + 9);

```

```

            pDC->SelectObject(&m_penColor);
            pDC->MoveTo(nXpos1, nYpos + 2);
            pDC->LineTo(nXpos2, nYpos + nHeight + 2);
            pDC->MoveTo(nXpos1, nYpos + 8);
            pDC->LineTo(nXpos2, nYpos + nHeight + 8);

```

```

            pDC->SelectObject(&m_penColorLight);
            pDC->MoveTo(nXpos1, nYpos + 3);
            pDC->LineTo(nXpos2, nYpos + nHeight + 3);
            pDC->MoveTo(nXpos1, nYpos + 7);

```

```

        pDC->LineTo(nXpos, nYpos + nHeight + 7);

        pDC->SelectObject(&m_penColorLighter);
        pDC->MoveTo(nXpos1, nYpos + 4);
        pDC->LineTo(nXpos2, nYpos + nHeight + 4);
        pDC->MoveTo(nXpos1, nYpos + 6);
        pDC->LineTo(nXpos2, nYpos + nHeight + 6);
    } // for the number of bands
} // if indeterminate
else
{
    if (nHeight > 3)
    {
        pDC->MoveTo(nLeft, nTop + 1);
        pDC->LineTo(nLeft, nTop);
        pDC->LineTo(nRight, nTop);
        pDC->MoveTo(nLeft + 1, nBottom - 2);
        pDC->LineTo(nLeft + 1, nTop + 1);
        pDC->MoveTo(nRight - 3, nBottom - 3);
        pDC->LineTo(nRight - 3, nTop + 1);
        pDC->SetPixel(nRight - 2, nTop + 1, m_crColor);

        pDC->SelectObject(&m_penColorLight);
        pDC->MoveTo(nLeft + 2, nBottom - 3);
        pDC->LineTo(nLeft + 2, nTop + 1);
        pDC->MoveTo(nRight - 4, nBottom - 3);
        pDC->LineTo(nRight - 4, nTop + 1);
        pDC->SetPixel(nLeft + 1, nTop + 1, m_crColorLight);
        pDC->SetPixel(nRight - 3, nTop + 1, m_crColorLight);

        pDC->SelectObject(&m_penColorLighter);
        pDC->MoveTo(nLeft + 3, nBottom - 3);
        pDC->LineTo(nLeft + 3, nTop + 1);
        pDC->MoveTo(nRight - 5, nBottom - 3);
        pDC->LineTo(nRight - 5, nTop + 1);
        pDC->SetPixel(nLeft + 2, nTop + 1, m_crColorLighter);
        pDC->SetPixel(nRight - 4, nTop + 1, m_crColorLighter);

        pDC->SelectObject(&m_penColorDark);
        pDC->MoveTo(nLeft, nBottom - 1);
        pDC->LineTo(nLeft, nTop + 1);
        pDC->MoveTo(nLeft + 2, nBottom - 2);
        pDC->LineTo(nRight - 2, nBottom - 2);
        pDC->LineTo(nRight - 2, nTop + 1);
        pDC->SetPixel(nRight - 1, nTop + 1, m_crColorDark);

        pDC->SelectObject(&m_penColorDarker);
        pDC->MoveTo(nLeft + 1, nBottom - 1);
        pDC->LineTo(nRight - 1, nBottom - 1);
        pDC->LineTo(nRight - 1, nTop + 1);
    }
}
else
{
    CBrush br(m_crColor);
    CBrush *pOldBrush = pDC->SelectObject(&br);
    pDC->SelectObject(&m_penColorDark);
    pDC->Rectangle(rect);
    pDC->SelectObject(pOldBrush);
}
} // if not indeterminate

pDC->SelectObject(pOldPen);
} // DrawVerticalBar

//-----
//
BOOL CMacProgressCtrl::OnEraseBkgnd(CDC* pDC)
//
// Return Value:    Nonzero if it erases the background; otherwise 0.
//
// Parameters      :    pDC - Specifies the device-context object.
//
// Remarks         :    The framework calls this member function when the
//                      CWnd object background needs erasing (for example,

```

```

// when (sized). It is called to prepare a validated
// region for painting.
//
{
    return TRUE;
} // OnEraseBkgnd

//-----
//
void CMacProgressCtrl::GetColors()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Calculates the lighter and darker colors, as well as
//                      the shadow colors.
//
{
    m_crColorLight = LightenColor(m_crColor, 51);
    m_crColorLighter = LightenColor(m_crColorLight, 51);
    m_crColorLightest = LightenColor(m_crColorLighter, 51);
    m_crColorDark = DarkenColor(m_crColor, 51);
    m_crColorDarker = DarkenColor(m_crColorDark, 51);
    m_crDkShadow = ::GetSysColor(COLOR_3DDKSHADOW);
    m_crLiteShadow = ::GetSysColor(COLOR_3DSHADOW);

    // Get a color halfway between COLOR_3DDKSHADOW and COLOR_3DSHADOW
    BYTE byRed3DDkShadow = GetRValue(m_crDkShadow);
    BYTE byRed3DLiteShadow = GetRValue(m_crLiteShadow);
    BYTE byGreen3DDkShadow = GetGValue(m_crDkShadow);
    BYTE byGreen3DLiteShadow = GetGValue(m_crLiteShadow);
    BYTE byBlue3DDkShadow = GetBValue(m_crDkShadow);
    BYTE byBlue3DLiteShadow = GetBValue(m_crLiteShadow);

    m_crShadow = RGB(byRed3DLiteShadow + ((byRed3DDkShadow - byRed3DLiteShadow) >> 1),
                    byGreen3DLiteShadow + ((byGreen3DDkShadow - byGreen3DLiteShadow) >> 1),
                    byBlue3DLiteShadow + ((byBlue3DDkShadow - byBlue3DLiteShadow) >> 1));
} // GetColors

//-----
//
void CMacProgressCtrl::SetColor(COLORREF crColor)
//
// Return Value:    None.
//
// Parameters      :    crColor - New color.
//
// Remarks         :    Sets the progress bar control's color. The lighter
//                      darker colors are recalculated, and the pens recreated.
//
{
    m_crColor = crColor;
    GetColors();
    CreatePens();
    RedrawWindow();
} // SetColor

//-----
//
COLORREF CMacProgressCtrl::GetColor()
//
// Return Value:    The current color.
//
// Parameters      :    None.
//
// Remarks         :    Returns the progress bar control's current color.
//
{
    return m_crColor;
} // GetColor

//-----
//

```

```

void CMacProgressCtrl::CreatePens()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Deletes the pen objects, if necessary, and creates them.
//
{
    DeletePens();

    m_penColorLight.CreatePen(PS_SOLID, 1, m_crColorLight);
    m_penColorLighter.CreatePen(PS_SOLID, 1, m_crColorLighter);
    m_penColor.CreatePen(PS_SOLID, 1, m_crColor);
    m_penColorDark.CreatePen(PS_SOLID, 1, m_crColorDark);
    m_penColorDarker.CreatePen(PS_SOLID, 1, m_crColorDarker);
    m_penDkShadow.CreatePen(PS_SOLID, 1, m_crDkShadow);
    m_penShadow.CreatePen(PS_SOLID, 1, m_crShadow);
    m_penLiteShadow.CreatePen(PS_SOLID, 1, m_crLiteShadow);
} // CreatePens

//-----
//
void CMacProgressCtrl::DeletePens()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Deletes the pen objects.
//
{
    if (m_penColorLight.m_hObject)
        m_penColorLight.DeleteObject();
    if (m_penColorLighter.m_hObject)
        m_penColorLighter.DeleteObject();
    if (m_penColor.m_hObject)
        m_penColor.DeleteObject();
    if (m_penColorDark.m_hObject)
        m_penColorDark.DeleteObject();
    if (m_penColorDarker.m_hObject)
        m_penColorDarker.DeleteObject();
    if (m_penDkShadow.m_hObject)
        m_penDkShadow.DeleteObject();
    if (m_penShadow.m_hObject)
        m_penShadow.DeleteObject();
    if (m_penLiteShadow.m_hObject)
        m_penLiteShadow.DeleteObject();
} // DeletePens

//-----
//
void CMacProgressCtrl::SetIndeterminate(BOOL bIndeterminate)
//
// Return Value:    None.
//
// Parameters      :    bIndeterminate - Specifies the indeterminate state.
//
// Remarks         :    Sets the indeterminate flag.
//
{
    m_bIndeterminate = bIndeterminate;

    if (m_bIndeterminate)
    {
        CRect rect;
        GetClientRect(rect);
        m_nIndOffset = 0;

        RedrawWindow();
        SetTimer(IDT_INDETERMINATE, 25, NULL);
    }
    else
    {

```

```

        KillTimer(IDT_INDETERMINATE);
        RedrawWindow();
    }
    // SetIndeterminate

//-----
//
BOOL CMacProgressCtrl::GetIndeterminate()
//
// Return Value:    m_bIndeterminate.
//
// Parameters      :    None.
//
// Remarks         :    Returns m_bIndeterminate.
//
{
    return m_bIndeterminate;
} // GetIndeterminate

//-----
//
void CMacProgressCtrl::OnTimer(UINT nIDEvent)
//
// Return Value:    None.
//
// Parameters      :    nIDEvent - Specifies the identifier of the timer.
//
// Remarks         :    The framework calls this member function after each
//                      interval specified in the SetTimer member function used
//                      to install a timer.
//
{
    // Increment the indeterminate bar offset and redraw the window.
    if (nIDEvent == IDT_INDETERMINATE)
    {
        KillTimer(nIDEvent);

        if (++m_nIndOffset > IND_BAND_WIDTH - 1)
            m_nIndOffset = 0;
        RedrawWindow();

        SetTimer(IDT_INDETERMINATE, 25, NULL);
    }
} // OnTimer

//=====
// OProgress Class
//=====
//=====
// Construction/Destruction
//=====

OPProgress::OPProgress()
{
}

OPProgress::~OPProgress()
{
}

/*****
 *
 *   Initialize progress control
 *   and insert it in the main frame window status bar
 *
 *****/
bool OPProgress::Initialize()
{
    CMDIFrameWnd* mf=(CMDIFrameWnd*)AfxGetMainWnd();
    if(!mf)
    {
        AfxMessageBox("Failed to create progress control");
    }
}

```

```

        return false;
    }
    CStatusBar* main_status_bar=(CStatusBar*)(mf->GetMessageBar());
    if(!main_status_bar)
    {
        AfxMessageBox("Failed to create progress control");
        return false;
    }
    RECT rc;    main_status_bar->GetItemRect(1,&rc);
    if(!Create(WCHILD | WS_VISIBLE, rc, main_status_bar, 1))
    {
        AfxMessageBox("Failed to create progress control");
        return false;
    }
    SetRange(0,100);    SetPos(0);    SetColor(RED);
    return true;
}

/*****
*
*   Show progress with optional message string
*   in the main frame window status bar
*
*****/
void OProgress::ShowProgress(int percent, char *info /*=NULL*/)
{
    if(GetSafeHwnd()==NULL) return;
    if(percent<=0)    percent=0;
    else if(percent>100)    percent=100;
    SetPos(percent);
    if(percent==0 || percent==100)    info="Ready";
    if(info)
    {
        CMDIFrameWnd* mf=(CMDIFrameWnd*)AfxGetMainWnd();
        if(!mf) return;
        CStatusBar* main_status_bar=(CStatusBar*)(mf->GetMessageBar());
        if(!main_status_bar)    return;
        main_status_bar->SetPaneText(0,info);
    }
}

```

```
// IEToolBar.h: interface for the IEToolBar class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_IEToolBar_H__467C453D_E943_11D3_977E_00105A21774F__INCLUDED_
#define AFX_IEToolBar_H__467C453D_E943_11D3_977E_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include <afxext.h>

class IEToolBar : public CToolBar
{
public:
    void        SetInsertedControlText(CString s);
    bool        TrackDropDownMenu(UINT buttonID, CWnd* pParent);
    bool        AttachDropDown(UINT buttonID, UINT menuID, UINT menuItemID);
    bool        MakeCStatic(UINT id);
    BOOL        CreateIE( CWnd* pParentWnd, UINT img_width,
                        UINT img_height, UINT resource );

    IEToolBar();
    ~IEToolBar();

private:
    UINT        m_ImageWidth, m_ImageHeight;
    UINT*        menuIDs;
    UINT*        menuItemIDs;
    CStatic*     m_Static;

    void        SetButtonsIE();
    bool        InsertControl(int ctrl_type, UINT nID, CString title="");
    CMenu*      GetSubMenuFromID(CMenu* menu, UINT id);
}

#endif // !defined(AFX_IEToolBar_H__467C453D_E943_11D3_977E_00105A21774F__INCLUDED_)
```

```
// IEToolBar.cpp: implementation of the IEToolBar class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "IEToolBar.h"

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

IEToolBar::IEToolBar()
{
    m_Static=NULL;
    menuIDs=NULL;
    menuItemIDs=NULL;
}

IEToolBar::~IEToolBar()
{
    if(m_Static) delete m_Static;
    if(menuIDs) delete [] menuIDs;
    if(menuItemIDs) delete [] menuItemIDs;
}

/*****
*
* Creating toolbars with IE style
*
*****/
BOOL IEToolBar::CreateIE(CWnd *pParentWnd, UINT img_width,
                        UINT img_height, UINT resource)
{
    m_Static=NULL; menuIDs=NULL; menuItemIDs=NULL;
    m_ImageWidth=img_width; m_ImageHeight=img_height;
    if(CToolBar::CreateEx(pParentWnd,TBSTYLE_FLAT,
        WS_CHILD | WS_VISIBLE | CBRS_ALIGN_TOP |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC)==FALSE)
    {
        AfxMessageBox("Failed to create toolbar");
        return FALSE;
    }
    if(LoadToolBar(resource)==FALSE)
    {
        AfxMessageBox("Failed to load toolbar");
        return FALSE;
    }
    SetButtonsIE();

    // TODO: Remove this if you don't want tool tips or a resizable toolbar
    SetBarStyle(GetBarStyle() | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);
    // Support for dropdown arrows
    GetToolBarCtrl().SetExtendedStyle(TBSTYLE_EX_DRAWDDARROWS);
    menuIDs = new UINT[GetCount()+5];
    menuItemIDs = new UINT[GetCount()+5];
    for(int i = 0; i < GetCount()+5; i++) menuItemIDs[i]=menuIDs[i]=0;

    return TRUE;
}

/*****
*
* Set IE-like flat buttons with text
*
*****/
void IEToolBar::SetButtonsIE()
{
    // Add text to each button
    int tlength=5;
    for(int i = 0; i < GetCount(); i++)
    {
        UINT id = GetItemID(i);
        CString s; if(!s.LoadString(id)) continue;
        int j = s.Find('\n');
        if(j < 0) continue;
        else s=s.Mid(j+1);
    }
}

```



```

        if(s.GetLength()>tlength) s=s.Left(tlength);
        SetButtonText(i,s);
    }
    // Resize buttons to include text
    CRect rect;
    GetItemRect(0,&rect);
    SetSizes(rect.Size(),CSize(m_ImageWidth,m_ImageHeight));
}
/*****
*
* Associate given id with CStatic control
*****/
bool IEToolBar::MakeCStatic(UINT id)
{
    return InsertControl(1,id,"0");
}
/*****
*
* Insert a control instead nID button
*****/
bool IEToolBar::InsertControl(int ctrl_type, UINT nID,
                              CString title /*="" */)
{
    DWORD dwStyle = WS_CHILD | WS_VISIBLE;
    CWnd* pCtrl=NULL;
    CRect rect;

    // Make sure the id is valid
    int index = CommandToIndex( nID ); if(index<0) return false;
    GetItemRect(index,rect); rect.left += 15;
    SetButtonInfo(index, nID, TBBS_SEPARATOR, rect.Width());

    // Insert the control
    switch(ctrl_type)
    {
    case 1: // CStatic
        if(m_Static) return false;
        m_Static = new CStatic(); if(!m_Static) return false;
        dwStyle |= SS_CENTER;
        if(!m_Static->Create(title, dwStyle, rect, this, nID))
        {
            delete m_Static; m_Static=NULL; return false;
        }
        pCtrl=m_Static;
        break;
    default: return false;
    }
    GetItemRect(index, &rect );
    pCtrl->SetWindowPos(0, rect.left, rect.top, 0, 0,
        SWP_NOZORDER | SWP_NOACTIVATE | SWP_NOSIZE | SWP_NOCOPYBITS );
    pCtrl->ShowWindow( SW_SHOW );
    return true;
}
/*****
*
* If a control was inserted into the toolbar
* set it text to a given string
*****/
void IEToolBar::SetInsertedControlText(CString s)
{
    if(m_Static) m_Static->SetWindowText(CString("\n")+s);
}
/*****
*
* Attach dropdwln arrow and menu to a given button
*****/

```

```

bool IEToolBar::AttachDropDown(UINT buttonID, UINT menuID, UINT menuItemID)
{
    // Make sure the id is valid
    int index = CommandToIndex(buttonID);
    if(index<0 || index>=GetCount()) return false;
    DWORD dwStyle = GetButtonStyle(index);
    dwStyle |= TBSTYLE_DROPDOWN;
    SetButtonStyle(index, dwStyle);
    menuIDs[index]=menuID;
    menuItemIDs[index]=menuItemID;
    return true;
}

/*****
*
*   Display dropdown menu
*
*****/
bool IEToolBar::TrackDropDownMenu(UINT buttonID, CWnd *pParent)
{
    // Make sure the id is valid
    int index = CommandToIndex(buttonID);
    if(index<0 || index>=GetCount()) return false;
    // Find menu ID
    UINT menuID=menuIDs[index];
    if(menuID==0) return true;    // no menu attached
    // Load and display popup menu
    CMenu menu;    menu.LoadMenu(menuID);
    CMenu* pPopup=GetSubmenuFromID(&menu,menuItemIDs[index]);

    if(!pPopup) return true;    // no such submenu
    CRect rc;
    this->SendMessage(TB_GETRECT, buttonID, (LPARAM)&rc);
    this->ClientToScreen(&rc);
    pPopup->TrackPopupMenu( TPM_LEFTALIGN | TPM_LEFTBUTTON | TPM_VERTICAL,
        rc.left, rc.bottom, pParent, &rc);
    return true;
}

/*****
*
*   Find submenu which contains given menu id
*
*****/
CMenu* IEToolBar::GetSubmenuFromID(CMenu* menu, UINT id)
{
    CMenu* sub;
    if(!menu) return NULL;
    UINT c = menu->GetMenuItemCount();
    if(c<=0) return NULL;
    for(UINT i=0; i<c; i++)
    {
        if(menu->GetMenuItemID(i)==id) return menu;
        sub = menu->GetSubMenu(i);
        if(!sub) continue;
        sub=GetSubmenuFromID(sub,id);
        if(sub) return sub;
    }
    return NULL;
}

```

```

// RayTracer.h: interface for the RayTracer class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_RAYTRACER_H__9741FB4F_8B17_11D3_9720_00105A21774F__INCLUDED_
#define AFX_RAYTRACER_H__9741FB4F_8B17_11D3_9720_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include "RTScreen.h"
#include "Volume.h"

class RayTracer
{
public:
    void SetBodyColor(int cmin, int cmax);
    void SetSurfaceLevel(int lev);
    void SetGranularity(int gr);
    void SetEyePosition(double x, double y, double z);
    void SetEyePositionOnSphere(double deg_hor, double deg_ver);
    bool SetVolumeScales(double sx, double sy, double sz);
    char r_error[64];

    double DoTracing();
    RayTracer(Volume* v, RTScreen* rts);
    virtual ~RayTracer();

private:
    signed char IS[16][16], AX, AY, AZ, r_OX, r_OY, r_OZ;
    int r_ALscale[4], r_N[4], Na[4], NX[14], IED[4], IL[4], P[4], TOP[4];
    int r_COLb, r_COLf, r_COLc, r_COLL, r_LE, r_HI, r_MAXX, r_MAXY, r_DIM, r_DIM1;
    int Fmax, Fmin, r_LEV;
    int r_RotationCode;
    long BD, IL12, IL13, IL23, DL1, DL2, PE, QE, AA, r_DIM2, r_DIM3;
    double r_Leye[4], LL[4], AP[4], B1, B2, FIL;
    double DOWN, CU1, CU2, CV1, CV2, CV3, CF1, CF2, CF3;
    double LLL, D1, D2, Fk;
    RTScreen* r_Screen;
    Volume* r_Volume;

    void EDGE();
    void INST (int p, int q, int u, int v);
    bool PICTURE (char how);
    char TEST (int w1, int w2, int w3, int w4);
    int Q_JUMP ();
    double FUNC(long x, long y, long z);
    double JUMP (double t);

    // inline functions
    inline void CHESS (int uu, int vv, int i1, int i2, int i3, int i4, int j1, int j2,
                      int j3, int j4, int j5, int j6, int j7, int j8 );
    inline void GURO2(char mask, long i1, long i2, long i3, long i4, long j1, long j2,
                     long j3, long j4, long j5, long j6, long j7, long j8, int ix, int iy);
    inline bool CUBESIGN(int a1, int a2, int a3, int a4, int a5, int a6, int a7, int a8);
    inline int INTENS (double a, double b, double c, double d, double t);
    inline int F(int x, int y, int z);
    inline int F_XpYpZp(int x, int y, int z);
    inline int F_XmYpZp(int x, int y, int z);
    inline int F_XpYmZp(int x, int y, int z);
    inline int F_XmYmZp(int x, int y, int z);
    inline int F_XpYpZm(int x, int y, int z);
    inline int F_XmYpZm(int x, int y, int z);
    inline int F_XpYmZm(int x, int y, int z);
    inline int F_XmYmZm(int x, int y, int z);
};

#endif // !defined(AFX_RAYTRACER_H__9741FB4F_8B17_11D3_9720_00105A21774F__INCLUDED_)

```

2

```
// RayTracer.cpp: implementation of the RayTracer class.
//
//
#include "RayTracer.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

// Construction/Destruction
//

RayTracer::RayTracer(Volume* v, RTScreen* rts)
{
    r_Volume=v; r_Screen=rts;
    SetVolumeScales(1,1,1);
    SetEyePosition(1,1,1);
    SetGranularity(4);
    SetSurfaceLevel(0);
    SetBodyColor(10,255);
}

RayTracer::~RayTracer() { }

int (RayTracer::*F_XYZ)(int x, int y, int z);

/*
 * Ray tracing; returns tracing time
 */
double RayTracer::DoTracing()
{
    int i,j,j0,j1,k,m,m1,m2,lef,mid,rig,idim,jdim;
    int jf0,jf1,jl0,jl1,jm0,jm1,jmin,jmax,p0,p1,p2,p3,jj1,jj2,jj5,jj6,wd[4][500];
    double u1,u2,v1,v2,v3,du,un1,un2,vn1,vn2,vn3,a,b,h1,h2,e1,e2,e3,e4;
    double x0,y0,z0,t,ddt,dt[4];
    double x[4],td[3][500];
    clock_t clock_start,clock_end;

    /* ***** RAY TRACING PARAMETERS */
    // Volume sizes in (x,y,z)
    r_N[1]=r_Volume->GetXSize();
    r_N[2]=r_Volume->GetYSize();
    r_N[3]=r_Volume->GetZSize();
    // int Axes numbers, to account for views from any octant
    r_OX=1; r_OY=2; r_OZ=3;
    // int Colors
    r_COLc=10; // non-body background
    r_COLL=5; // color to draw the volume frame
    r_MAXX=800; r_MAXY=700; // max screen sizes for traced picture
    r_LE=r_MAXX-10; r_HI=r_MAXY-10; // actual screen sizes for traced picture

    /* ***** Do we need scaling ? */
    r_Volume->GetMinMax(Fmin,Fmax);
    PE=Fmax-Fmin;
    if(r_LEV<Fmin || r_LEV>Fmax)
    {
        sprintf(r_error,"Chosen level must be in [%d,%d] interval",Fmin,Fmax);
        return -1.0;
    }
    if(PE>500)
    {
        Fk=(double)(500./PE);
        r_LEV=(int)(r_LEV*Fk+0.5);
        sprintf(r_error,"Data needs to be scaled, current range is [%d,%d]",Fmin,Fmax);
        return -1.0; // Need F(i,j,k) -> (F[i][j][k]-Fmin)*Fk
    }
}
```

```

else    Fk=1.;
FIL=(double)(10000*sqrt( 5./((Fmax-Fmin+1)) ));

/***** Compute picture parameters */
if(!PICTURE ('Y')) return -1.0;
AX=1;    AY=2;    AZ=3;
u1=CU1;    u2=CU2;
v1=CV1;    v2=CV2;    v3=CV3;
du=AP[1];    un1=AP[2];    un2=AP[3];
vn1=CF1;    vn2=CF2;    vn3=CF3;
m1=r_DIM1;    m2=r_DIM2;

/***** Accomodate different quadrants */
r_RotationCode=0;
if (r_Leye[1]<0)    r_RotationCode += 1;
if (r_Leye[2]<0)    r_RotationCode += 10;
if (r_Leye[3]<0)    r_RotationCode += 100;
switch(r_RotationCode)
{
case 001:
    F_XYZ=(this->F_XpYpZm);
    break;
case 010:
    F_XYZ=(this->F_XpYmZp);
    break;
case 011:
    F_XYZ=(this->F_XpYmZm);
    break;
case 100:
    F_XYZ=(this->F_XmYpZp);
    break;
case 101:
    F_XYZ=(this->F_XmYpZm);
    break;
case 110:
    F_XYZ=(this->F_XmYmZp);
    break;
case 111:
    F_XYZ=(this->F_XmYmZm);
    break;
default:
    F_XYZ=(this->F_XpYpZp);
    break;
}
if(AZ<0) { if( (AX==2) || (AX==-2) ) { p1=r_N[2]; p2=r_N[1]; }
           else { p1=r_N[1]; p2=r_N[2]; }
for(k=0;k<=(r_N[3]-1)/2;k++) { p3=r_N[3]-k;
    for(i=0;i<=p1;i++) { for(j=0;j<=p2;j++) {
        p0=F(i,j,k); F(i,j,k)=F[i][j][p3]; F[i][j][p3]=p0; } }
    } // next k
} // end if
switch(AX) {
case -1: p1=r_N[1]/2;
for(k=0;k<=r_N[3];k++) { for(i=0;i<=p1;i++) {
    for(j=0;j<=r_N[2];j++) {
        p0=F(i,j,k); F(i,j,k)=F[r_N[1]-i][r_N[2]-j][k]; F[r_N[1]-i][r_N[2]-j][k]=p0; }
    } // next k
}
break;
case 2: p1=max(r_N[1],r_N[2]);
for(k=0;k<=r_N[3];k++) {
    for(i=0;i<=(r_N[2]-1)/2;i++) { for(j=0;j<=r_N[1];j++) {
        p0=F(i,j,k); F(i,j,k)=F[r_N[2]-i][j][k]; F[r_N[2]-i][j][k]=p0; } }
    for(i=0;i<=p1;i++) { for(j=0;j<i;j++) {
        p0=F(i,j,k); F(i,j,k)=F[j][i][k]; F[j][i][k]=p0; } }
    } // next k
}
break;
case -2: p1=max(r_N[1],r_N[2]);
for(k=0;k<=r_N[3];k++) {
    for(i=0;i<=r_N[2];i++) { for(j=0;j<=(r_N[1]-1)/2;j++) {
        p0=F(i,j,k); F(i,j,k)=F[i][r_N[1]-j][k]; F[i][r_N[1]-j][k]=p0; } }
    for(i=0;i<=p1;i++) { for(j=0;j<i;j++) {

```

```

    p0=F(i,j,k); F(i,j,k)=F(j)[k]; F[j][i][k]=p0; } }
    } // next k
break;
} // end switch
*/
/***** DRAW AND FILL POLIGON */
/!!! Draw frame ?
/*
if (r_COLc!=0) {setfillstyle(SOLID_FILL,r_COLc) ; fillpoly(6,NX);}
setcolor (r_COLl);drawpoly(7,NX);
line(-un1,r_HI+vn1+vn2,NX[0],NX[1]);
line(-un1,r_HI+vn1+vn2,NX[4],NX[5]);
line(-un1,r_HI+vn1+vn2,NX[8],NX[9]);
*/
clock_start=clock();

/***** PREPARATIONS: */
h1=v1/u1 ; h2=v2/u2 ;
IL12=IL[1]; IL13=IL[1]; IL23=IL[2];
IL12*=IL[2]; IL13*=IL[3]; IL23*=IL[3];
r_DIM1=r_DIM-1; r_DIM2=r_DIM*r_DIM ; r_DIM3=r_DIM2*r_DIM ;
for(i=1;i<4;i++) { dt[i]=r_Leye[i]*r_Leye[i]; AP[i]=(double)(0.5-r_N[i]*(double)(IL[i])); }
DL1=-2*IL[1]; DL2=-2*IL[2];
D1=(double)(0.25/IL12); D2=(double)(-0.5*D1/IL[3]);
a=(double)(__min( 30000 , 1000000000./(13.*r_DIM3+43*r_DIM2+10*r_DIM) ));
i=(int)(a/(r_COLb+r_COLf));
DOWN=2*du*LLL ; B2=i*r_COLf*DOWN ; DOWN*=DOWN ;
B1=(double)(i*r_COLb+0.5) ; BD=4*i*(long)(r_DIM3) ;
b=(double)(Na[1]+Na[2]+Na[3]) ; a=(double)(0.001*du*m1/2) ; du*=0.999 ;
CF1=Na[1]*(u1*u1+v1*v1)+Na[2]*v1*v2-b*r_Leye[1]+u1*a ; CF1=(Na[1]-CF1)*LL[1]*LLL;
CF2=Na[1]*(u1*u2+v1*v2)+Na[2]*v2*v2-b*r_Leye[2]+u2*a ; CF2=(Na[2]-CF2)*LL[2]*LLL;
CF3=v3*(Na[1]*v1+Na[2]*v2)-b*r_Leye[3]; CF3=(Na[3]-CF3)*LL[3]*LLL;
CU1=-LL[1]*u1*du*LLL/r_DIM; CU2=-LL[2]*u2*du*LLL/r_DIM; CV1=-LL[1]*v1*du*LLL/r_DIM;
CV2=-LL[2]*v2*du*LLL/r_DIM; CV3=-LL[3]*v3*du*LLL/r_DIM;
e1=(vn3-vn2)/r_DIM ; e2=(un1*h2-vn1-vn2+vn3)/r_DIM;
e3=-vn2/r_DIM ; e4=-un2*h1/r_DIM;
x0=CF1 ; y0=CF2 ; z0=CF3 ;
u1=CU1*r_DIM ; u2=CU2*r_DIM ; v1=CV1*r_DIM ; v2=CV2*r_DIM ; v3=CV3*r_DIM ;
for(i=1;i<4;i++) r_Leye[i]/=r_ALscale[i];
/***** MAIN LOOP: */
rfg=0 ;
for (i=0,idim=0;i<=m1;i++,idim+=r_DIM) {
    printf("%3d%",(int)(100*i/m1));
    j1=(int)(__min(e1,e2)+0.99) ; j0=(int)(__max(e3,e4)) ;
    e1+=h1 ; e2+=h2 ; e3+=h2 ; e4+=h1 ;
    x[1]=x0+j0*v1 ; x[2]=y0+j0*v2 ; x[3]=z0+j0*v3 ;
    for (j=j0;j<=j1;j++) {
        /* GO INTO CUBE: */
        t=__max(__max(x[1],x[2]),x[3]); ddt=0.;
        for(k=1;k<4;k++) {
            if(t==x[k]) { NX[k]=r_N[k]; P[k]=-IL[k]; IED[k]=k; TOP[k]=r_N[k]-1; }
            else { a=AP[k]+x[k]-t; if(a<0.5) { td[rig][j]=-1; goto nextj; }
                AA=(long)a; a=-AA+0.5; NX[k]=AA/(-IL[k]); P[k]=AA+(long)(NX[k])*IL[k];
                if(P[k]==0) { if(NX[k]==0) { ddt+=(a-1)*dt[k]; P[k]=1;
                    IED[k]=0; TOP[k]=0; }
                    else { ddt+=a*dt[k]; P[k]=-IL[k];
                        IED[k]=k; TOP[k]=NX[k]-1; }
                }
            else { IED[k]=0; TOP[k]=NX[k]; ddt+=a*dt[k]; }
        }
        /* next k */
        td[rig][j]=JUMP(t+ddt); /* WE'VE FOUND THE BODY */
    }
    nextj: x[1]+=v1 ; x[2]+=v2 ; x[3]+=v3 ;
    } /* next j */

    for(j=0;j<j0;j++) td[rig][j]=-1;
    for(j=j1+1;j<=m2;j++) td[rig][j]=-1;
/* END OF FINDING ROOTS,LET'S DRAW: */
if(i==0) {j11=j1; j10=j0; rig=1; mid=0; goto nexti; }

```

```

if(i==1) {
    if(jl0<=0)
        wd[0][0]=INTENS(-1,td[0][1],td[1][0],-1,td[0][0]);
    else
        wd[0][jl0]=INTENS(-1,td[0][jl0+1],td[1][jl0],td[0][jl0-1],td[0][jl0]);
        /* putpixel(0,r_HI-jl0*r_DIM,wd[0][jl0]); */
    for(j=jl0+1,jdim=r_HI-(jl0+1)*r_DIM;j<=j11;j++,jdim-=r_DIM) { if(j<m2) t=td[0][j+1];
        else t=-1;
        wd[0][j]=INTENS(-1,t,td[1][j],td[0][j-1],td[0][j]);
        /* putpixel(0,jdim,wd[0][j]); */
    }
    for(j=0;j<jl0;j++) { wd[0][j]=0; } for(j=j11+1;j<=m2;j++) { wd[0][j]=0; }

    jm0=j0; jml=j1; lef=0; mid=1; rig=2; p0=0;
    goto nexti;
} /* end of case i==1 */

if(i==2) {
    if(jm0<=0)
        wd[1][0]=INTENS(td[lef][0],td[mid][1],td[rig][0],-1,td[mid][0]);
    else
        wd[1][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],td[rig][jm0],
            td[mid][jm0-1],td[mid][jm0]);
        /* putpixel(r_DIM,r_HI-jm0*r_DIM,wd[1][jm0]); */
    for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jml;j++,jdim-=r_DIM) {
        if(j<m2) t=td[mid][j+1];
        else t=-1;
        wd[1][j]=INTENS(td[lef][j],t,td[rig][j],td[mid][j-1],td[mid][j]);
        /* putpixel(r_DIM,jdim,wd[1][j]); */
    } /* next j */
    for(j=0;j<jm0;j++) { wd[1][j]=0; } for(j=jml+1;j<=m2;j++) { wd[1][j]=0; }
    jf0=jl0; jf1=j11; jl0=jm0; j11=jml; jm0=j0; jml=j1;
    m=lef; lef=mid; mid=rig; rig=m;
    p1=1; goto nexti;
} /* end of case i==2 */

if(i==3) {
    if(jm0<=0)
        wd[2][0]=INTENS(td[lef][0],td[mid][1],td[rig][0],-1,td[mid][0]);
    else
        wd[2][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],td[rig][jm0],
            td[mid][jm0-1],td[mid][jm0]);
        /* putpixel(idim-r_DIM,r_HI-jm0*r_DIM,wd[2][jm0]); */
    for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jml;j++,jdim-=r_DIM) {
        if(j<m2) t=td[mid][j+1];
        else t=-1;
        wd[2][j]=INTENS(td[lef][j],t,td[rig][j],td[mid][j-1],td[mid][j]);
        /* putpixel(idim-r_DIM,jdim,wd[2][j]); */
    }
    for(j=0;j<jm0;j++) { wd[2][j]=0; } for(j=jml+1;j<=m2;j++) { wd[2][j]=0; }
    jmin=__min(jf0,jl0); jmax=__max(jf1,j11);
    for(j=jmin,jdim=jmin*r_DIM;j<=jmax-1;j++,jdim+=r_DIM) {
        if(j==0) {jj5=0;jj6=0;} else {jj5=wd[1][j-1];jj6=wd[0][j-1];}
        if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[0][j+2];jj2=wd[1][j+2];}

    switch(TEST(wd[0][j],wd[1][j],wd[0][j+1],wd[1][j+1])) {
        case '?':
            CHESS(0,jdim,wd[0][j],wd[1][j],wd[0][j+1],wd[1][j+1],
                jj1,jj2,wd[2][j+1],wd[2][j],jj5,jj6,0,0);
            break;
        case '1':
            GURO2('-',wd[0][j],wd[1][j],wd[0][j+1],wd[1][j+1],
                jj1,jj2,wd[2][j+1],wd[2][j],jj5,jj6,0,0,0,jdim);
            break;
    } /* next j */
    jf0=jl0; jf1=j11; jl0=jm0; j11=jml; jm0=j0; jml=j1;
    m=lef; lef=mid; mid=rig; rig=m;
    p2=2; p3=3; goto nexti;
} /* end of case i==3 */

else { /* i>3 */
    if(jm0<=0)

```



```

wd[p3][0]=INTENS(td[lef][0],td[mid][1],td[rig][0],-1,td[mid][0]);
else
wd[p3][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],td[rig][jm0],
    td[mid][jm0-1],td[mid][jm0]);
/* putpixel(idim-r_DIM,r_HI-jmin*r_DIM,wd[p3][jm0]); */
for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jm1;j++,jdim-=r_DIM) {
    if(j<m2) t=td[mid][j+1];
    else t=-1;
    wd[p3][j]=INTENS(td[lef][j],t,td[rig][j],td[mid][j-1],td[mid][j]);
    /* putpixel(idim-r_DIM,jdim,wd[p3][j]); */
}
for(j=0;j<jm0;j++) { wd[p3][j]=0; } for(j=jm1+1;j<=m2;j++) { wd[p3][j]=0; }
jmin=__min(jf0,jl0) ; jmax=__max(jf1,jl1);
for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
    if(j==0) {jj5=0;jj6=0;} else {jj5=wd[p2][j-1];jj6=wd[p1][j-1];}
    if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[p1][j+2];jj2=wd[p2][j+2];}

switch(TEST(wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1])) {
    case '?':
        CHESS((i-3)*r_DIM,jdim,wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
            jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],wd[p0][j+1]);
        break ;
    case '1':
        GURO2('-',wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
            jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],
            wd[p0][j+1],(i-3)*r_DIM,jdim);
        break ;
}
} /* next j */
jf0=jl0 ; jf1=jl1 ; jl0=jm0 ; jl1=jm1 ; jm0=j0 ; jm1=j1 ;
m=lef; lef=mid; mid=rig; rig=m;
m=p0 ; p0=p1 ; p1=p2 ; p2=p3 ; p3=m ;
} /* end of case i>3 */

if(i==m1) {
    if(jm0<=0)
        wd[p3][0]=INTENS(td[lef][0],td[mid][1],-1,-1,td[mid][0]);
    else
        wd[p3][jm0]=INTENS(td[lef][jm0],td[mid][jm0+1],-1,td[mid][jm0-1],
            td[mid][jm0]);
    /* putpixel(idim,r_HI-jmin*r_DIM,wd[p3][jm0]); */
    for(j=jm0+1,jdim=r_HI-(jm0+1)*r_DIM;j<=jm1;j++,jdim-=r_DIM) {
        if(j<m2) t=td[mid][j+1];
        else t=-1;
        wd[p3][j]=INTENS(td[lef][j],t,-1,td[mid][j-1],td[mid][j]);
        /* putpixel(idim,jdim,wd[p3][j]); */
    }
    for(j=0;j<jm0;j++) { wd[p3][j]=0; } for(j=jm1+1;j<=m2;j++) { wd[p3][j]=0; }
    jmin=__min(jf0,jl0) ; jmax=__max(jf1,jl1);
    for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
        if(j==0) {jj5=0;jj6=0;} else {jj5=wd[p3][j-1];jj6=wd[p2][j-1];}
        if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[p1][j+2];jj2=wd[p2][j+2];}

switch(TEST(wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1])) {
    case '?':
        CHESS((m1-2)*r_DIM,jdim,wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
            jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],wd[p0][j+1]);
        break ;
    case '1':
        GURO2('-',wd[p1][j],wd[p2][j],wd[p1][j+1],wd[p2][j+1],
            jj1,jj2,wd[p3][j+1],wd[p3][j],jj5,jj6,wd[p0][j],
            wd[p0][j+1],(m1-2)*r_DIM,jdim);
        break ;
}
} /* next j */

jmin=__min(jm0,j0) ; jmax=__max(jm1,j1);
for(j=jmin,jdim=jmin*r_DIM; j<=jmax-1; j++,jdim+=r_DIM) {
    if(j==0) {jj5=0;jj6=0;} else {jj5=wd[p3][j-1];jj6=wd[p2][j-1];}
    if(j==m2-1) {jj1=0;jj2=0;} else {jj1=wd[p2][j+2];jj2=wd[p3][j+2];}

switch(TEST(wd[p2][j],wd[p3][j],wd[p2][j+1],wd[p3][j+1])) {
    case '?':

```

```

        CHESS((m1-1)*r_DIM,jdim,p2[j],wd[p3][j],wd[p2][j+1],wd[p3][j+1],
        jj1,jj2,0,0,jj5,jj6,wd[p1][j],wd[p1][j+1]);
        break;
    case '1':
        GURO2('-',wd[p2][j],wd[p3][j],wd[p2][j+1],wd[p3][j+1],
        jj1,jj2,0,0,jj5,jj6,wd[p1][j],wd[p1][j+1],(m1-1)*r_DIM,jdim);
        break;
    }
} /* next j */
} /* end of case i==m1 */

nexti: x0+=u1; y0+=u2;
} /* next i */
if(r_COL1!=0) { //!! Frame ??
    /*
    setcolor (r_COL1); setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    line (un2,r_HI-vn3,un2,r_HI);
    line (un2,r_HI-vn3,0,r_HI+vn2-vn3);
    line (un2,r_HI-vn3,NX[8],NX[7]);
    */
}

// Finish
for(i=1;i<4;i++) r_Leye[i]*=r_ALscale[i];
clock_end=clock();
return (double)(clock_end-clock_start)/CLOCKS_PER_SEC;
}

/*****
* TEST: test surface presence inside a square
*****/
char RayTracer::TEST(int w1, int w2, int w3, int w4)
{
    if(w1<=0)
    {
        if( (w2<=0) && (w3<=0) && (w4<=0) ) return ('o');
        IS[0][0]=-1;
        return ('?');
    }
    else
    {
        if( (w2>0) && (w3>0) && (w4>0) ) return('l');
        IS[0][0]=1;
        return('?');
    }
}

/*****
*
* Initialize picture parameters
*
*****/
bool RayTracer::PICTURE(char how)
{
    int i,m1,m2,nx[14];
    double xa[4],a,b,c,u1,u2,v1,v2,v3,du,un1,un2,vn1,vn2,vn3;

    for(i=1;i<4;i++) Na[i]=r_N[i]*r_ALscale[i];
    if(how=='Y')
    {
        for(i=1;i<4;i++) {xa[i]=fabs(r_ALscale[i]/r_Leye[i]);}
        b=__max(xa[2],xa[3]);
        a=__min( FIL/sqrt( __max(xa[1]*b,xa[2]*xa[3]) ), 15000./__max(xa[1],b) );
        for(i=1;i<4;i++)
        {
            IL[i]=-(int)__max(0.5+a*xa[i],50);
            r_Leye[i]=(100.*r_ALscale[i])/IL[i];
        }
    }
    else
    {

```

```

}

/*****
 *
 *   Update Ruler pop-up menu
 *
 *****/
void Ruler::UpdatePopupMenu(CMenu *pop)
{
    GetLength();
    if(r_pix_spacingX<0.0)
    {
        pop->EnableMenuItem(ID_RULER_MM,MF_GRAYED);
        pop->EnableMenuItem(ID_RULER_CM,MF_GRAYED);
        pop->EnableMenuItem(ID_RULER_IN,MF_GRAYED);
        pop->CheckMenuItem (ID_RULER_PIXELS,MF_CHECKED);
    }
    else
    {
        if(r_scale=="mm") pop->CheckMenuItem(ID_RULER_MM,MF_CHECKED );
        else if (r_scale=="cm") pop->CheckMenuItem(ID_RULER_CM,MF_CHECKED );
        else if (r_scale=="in") pop->CheckMenuItem(ID_RULER_IN,MF_CHECKED );
        else      pop->CheckMenuItem(ID_RULER_PIXELS,MF_CHECKED );
    }

    CString value;
    value.Format("%.2lf ",r_length);
    pop->ModifyMenu(ID_RULER_VALUE,MF_BYCOMMAND,ID_RULER_VALUE,value+r_scale);
}

/*****
 *
 *   Compute distance in pixels
 *
 *****/
void Ruler::GetLength()
{
    r_size=r_end-r_start;
    r_pix_length=_hypot(r_size.cx,r_size.cy);
    if(r_scale=="pixels")
        r_length=r_pix_length;
    else
        r_length=r_scale_coeff*_hypot(r_size.cx*r_pix_spacingX,
                                       r_size.cy*r_pix_spacingY)/(*r_zoom);
}

/*****
 *
 *   Increment or decrement the number of tick marks if d!=0
 *
 *****/
void Ruler::ChangeTicksAndStyle(CDC *pDC, int d)
{
    if(!r_active) return;
    if(d!=0)      // change number of tick marks
    {
        int newticks=r_ticks+d;
        if(newticks<0)      newticks=0;
        else if(newticks>10) newticks=10;
        if(newticks==r_ticks) return;

        if(r_undo) Draw(pDC);
        r_ticks=newticks;
        Draw(pDC);
    }
}

/*****
 *
 *   Set distance scale
 *
 *****/
void Ruler::SetScale(int code)

```

[illegible]

```

// Selector.h: interface for Selector class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_SELECTOR_H__4E54B653_BCCC_11D2_95F9_00105A21774F__INCLUDED_
#define AFX_SELECTOR_H__4E54B653_BCCC_11D2_95F9_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define SEL_RECTANGLE 0
#define SEL_ELLIPSE 1

class Selector
{
public:
    bool m_undo, m_active;
    int m_shape;

    void Redraw(CDC *pDC, const CPoint& center,
               const CPoint& vertex, const CRect& client);
    void Redraw(CDC *pDC, const CPoint& center);
    void Remove(CDC *pDC);
    void Initialize(CDC* pDC, const CPoint& center, double* zoom,
                  double scaleX, double scaleY, int shape=0);
    CString toString();
    CRect GetRect();
    Selector();
    virtual ~Selector();

private:
    double m_scaleX, m_scaleY, m_area;
    double * m_zoom;
    CRect m_RectSel;

    void Draw(CDC* pDC);
    void Clean();
};

#endif // !defined(AFX_SELECTOR_H__4E54B653_BCCC_11D2_95F9_00105A21774F__INCLUDED_)

```

```
// Selector.cpp: implementation of the Selector class.
```

```
//
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
#include "stdafx.h"
#include "DCM.h"
#include "Selector.h"
```

```
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
Selector::Selector()
```

```
{
    m_shape=SEL_RECTANGLE;
    Clean();
}
```

```
Selector::~Selector()
```

```
{
    ;
}
```

```
*****
Functions to Draw and Update Selector rectangle
```

```
*****/
void Selector::Initialize(CDC *pDC, const CPoint& center, double* zoom,
    double scaleX, double scaleY, int shape)
```

```
{
    m_active=true;
    m_undo=true;
    m_shape=shape;
    if(m_shape!=SEL_ELLIPSE)    m_shape=SEL_RECTANGLE;
    if(scaleX!=0.0)
    {
        m_scaleX=m_scaleY=scaleX;
        if(scaleY!=0.0) m_scaleY=scaleY;
    }
    if (zoom)    m_zoom=zoom;
    m_RectSel=CRect(center.x-128, center.y-128,center.x+127, center.y+127);
    Draw(pDC);
}
```

```
void Selector::Draw(CDC *pDC)
```

```
{
    int    dmode;
    CPen* old_pen = pDC->SelectObject(&theApp.app_Pen);
    switch (m_shape)
    {
    case SEL_RECTANGLE:
        dmode=SetROP2(pDC->m_hDC, R2_NOT);
        pDC->MoveTo(m_RectSel.TopLeft());
        pDC->LineTo(m_RectSel.right,m_RectSel.top);
        pDC->LineTo(m_RectSel.right,m_RectSel.bottom);
        pDC->LineTo(m_RectSel.left,m_RectSel.bottom);
        pDC->LineTo(m_RectSel.TopLeft());
        SetROP2(pDC->m_hDC, dmode);
        break;
    case SEL_ELLIPSE:
        dmode=SetROP2(pDC->m_hDC, R2_NOT);
        pDC->Arc(m_RectSel,m_RectSel.TopLeft(),m_RectSel.TopLeft());
        SetROP2(pDC->m_hDC, dmode);
        break;
    }
}
```

```

    pDC->SelectObject(old_pen);
}

void Selector::Remove(CDC *pDC)
{
    if(m_undo) Draw(pDC);
    Clean();
}

void Selector::Redraw(CDC *pDC, const CPoint& center,
                     const CPoint& vertex, const CRect& client)    // resize
{
    if(!m_active) return;
    if(m_undo) Draw(pDC); // undo old selection
    CPoint z=vertex-center;
    int a=(abs(z.x)); if(a<16) a=16;
    int b=(abs(z.y)); if(b<16) b=16;
    m_RectSel=CRect(center.x-a, center.y-b,center.x+a, center.y+b);
    m_RectSel.IntersectRect(m_RectSel, client);
    m_RectSel.NormalizeRect();
    if(m_RectSel.IsRectEmpty()) return;
    Draw(pDC); // show new selection
    m_undo=true;
}

void Selector::Redraw(CDC *pDC, const CPoint& center)    // move
{
    if(!m_active) return;
    if(m_undo) Draw(pDC); // undo old selection CRect
    m_RectSel.OffsetRect(-m_RectSel.CenterPoint()+center);
    Draw(pDC); // show new selection
    m_undo=true;
}

/*****
 *
 * Reset selector
 *
 *****/
void Selector::Clean()
{
    m_active=false;
    m_undo=false;
    m_scaleX=m_scaleY=-1.0;
    m_area=0.0;
    double x=1.0; m_zoom = &x;
    m_RectSel=CRect(0,0,1,1);
}

/*****
 *
 * Return Selector region
 *
 *****/
CRect Selector::GetRect()
{
    return m_RectSel;
}

/*****
 *
 * Return Selector info
 *
 *****/
CString Selector::toString()
{
    CString info;
    CString units="pixels";
    double dx=m_RectSel.Width();
    double dy=m_RectSel.Height();
    if(m_scaleX>0 && (*m_zoom)>0)
    {
        units="mm";
        dx = m_scaleX*dx/(*m_zoom);
        dy = m_scaleY*dy/(*m_zoom);
    }
    if(m_shape==SEL_RECTANGLE) m_area=4.0*dx*dy;
}

```





```

#ifndef AFX_SOUNDDIALOG_H__750A3131_C0BF_11D2_9601_00105A21774F__INCLUDED_
#define AFX_SOUNDDIALOG_H__750A3131_C0BF_11D2_9601_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SoundDialog.h : header file
//

#include <mmsystem.h>

////////////////////////////////////
// SoundDialog dialog

class SoundDialog : public CDialog
{
// Construction
public:
    CString GetWavFileName();
    SoundDialog(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(SoundDialog)
    enum { IDD = IDD_SOUND_DIALOG };
        // NOTE: the ClassWizard will add data members here
    }}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(SoundDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    virtual BOOL OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult);
    }}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(SoundDialog)
    afx_msg void OnSoundPlay();
    virtual BOOL OnInitDialog();
    afx_msg void OnSoundRecord();
    afx_msg void OnSoundStop();
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    bool MakeFormatList();
    MMRESULT IsFormatSupported(LPWAVEFORMATEX pwfx, UINT uDeviceID);
    bool SetWaveFormat();
    bool CloseMCI();
    bool SaveToFile();
    bool m_Opened;
    MCI_OPEN_PARMS m_Device;
    CString m_info;
    CString ErrorMCI(DWORD e, CString intro=CString(""));
    bool OpenMCI();
    CString m_FileName;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SOUNDDIALOG_H__750A3131_C0BF_11D2_9601_00105A21774F__INCLUDED_)

```

```

// SoundDialog.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "SoundDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

// SoundDialog.cpp : implementation file
//
#include <mmreg.h>
#include <msacm.h>

////////////////////////////////////
// SoundDialog dialog

SoundDialog::SoundDialog(CWnd* pParent /*=NULL*/)
: CDialog(SoundDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(SoundDialog)
    // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void SoundDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(SoundDialog)
    // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(SoundDialog, CDialog)
    //{{AFX_MSG_MAP(SoundDialog)
    ON_BN_CLICKED(IDC_SOUND_PLAY, OnSoundPlay)
    ON_BN_CLICKED(IDC_SOUND_RECORD, OnSoundRecord)
    ON_BN_CLICKED(IDC_SOUND_STOP, OnSoundStop)
    ON_MESSAGE(MM_MCINOTIFY, OnNotify)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// SoundDialog message handlers

/*****
 *
 *   Play *.wav file
 *
 *****/
void SoundDialog::OnSoundPlay()
{
    PlaySound(m_FileName, NULL, SND_FILENAME);
}

/*****
 *
 *   Initialize dialog
 *
 *****/
BOOL SoundDialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_Opened=false;
    m_FileName=CString("sound.wav");
}

```

```

    return TRUE;
}

/*****
 *
 *   Record sound
 *
 *****/
void SoundDialog::OnSoundRecord()
{
    DWORD dwReturn;

    // Open a waveform-audio device with a new file for recording.
    if(!OpenMCI()) return;

    // Set recording parameters
    //SetWaveFormat();

    MCI_RECORD_PARMS mciRecordParms;
    // Record
    mciRecordParms.dwFrom = 0;
    mciRecordParms.dwTo = 1000;
    mciRecordParms.dwCallback = (DWORD)(this->GetSafeHwnd());
    Beep(900,100);
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_RECORD,
        MCI_FROM | MCI_NOTIFY, (DWORD)(LPVOID) &mciRecordParms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Recording error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
        return;
    }
    return;

    /*****
 *
 *   Stop recording/playing
 *
 *****/
void SoundDialog::OnSoundStop()
{
    DWORD dwReturn;
    MCI_GENERIC_PARMS genericParms;
    genericParms.dwCallback = (DWORD)this->GetSafeHwnd();
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_STOP,MCI_WAIT, (DWORD)(LPVOID) &genericPar
        ms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Stop error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
    }

    // Save and close MCI
    Beep(500,100);
    SaveToFile();
    CloseMCI();
}

/*****
 *
 *   Catch MM_MCINOTIFY
 *
 *****/
BOOL SoundDialog::OnNotify(WPARAM wParam, LPARAM lParam, LRESULT* pResult)
{
    //AfxMessageBox("On Notified");
    return TRUE;
}

/*****
 *
 *   Open MCI device
 *
 *****/

```

```

*****
bool SoundDialog::OpenMCI()
{
    DWORD dwReturn;

    // Open a waveform-audio device with a new file for recording.
    m_Device.lpstrDeviceType = "waveaudio";
    m_Device.lpstrElementName = "";
    if (dwReturn = mciSendCommand(0, MCI_OPEN, MCI_OPEN_ELEMENT | MCI_OPEN_TYPE,
        (DWORD)(LPVOID) &m_Device))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Open device error: "));
        m_Opened=false;
        return false;
    }
    m_Opened=true;
    return true;
}

/*****
*
*   Report MCI error
*
*****/
CString SoundDialog::ErrorMCI(DWORD e, CString intro)
{
    CString info;
    char ebuffer[200];
    if(! mciGetErrorString(e, ebuffer, 200) ) info.Format("Unknown error");
    else info=intro+CString(ebuffer);
    info.TrimRight();
    return info;
}

/*****
*
*   Save MCI recording into a file m_FileName
*
*****/
bool SoundDialog::SavetoFile()
{
    DWORD dwReturn;
    MCI_SAVE_PARMS mciSaveParms;
    mciSaveParms.lpfilename = m_FileName;
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_SAVE,
        MCI_SAVE_FILE | MCI_WAIT, (DWORD)(LPVOID) &mciSaveParms))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Save file error: "));
        return false;
    }
    return true;
}

/*****
*
*   Close MCI device
*
*****/
bool SoundDialog::CloseMCI()
{
    DWORD dwReturn;
    if (dwReturn = mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Close error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
    }
    return true;
    m_Opened=false;
}

```

```

}

bool SoundDialog::SetWaveFormat()
{
    /*
    DWORD dwReturn;
    MCI_WAVE_SET_PARMS mwspWaveFormParameters;
    mwspWaveFormParameters.wFormatTag=0x0002; //WAVE_FORMAT_DSPGROUP_TRUESPEECH ; //WAVE_FORMAT_AD
PCM ;
    if(dwReturn=mciSendCommand (m_Device.wDeviceID,MCI_SET,
        MCI_WAIT | MCI_WAVE_SET_FORMATTAG ,
        (DWORD)(LPVOID)&mwspWaveFormParameters))
    {
        AfxMessageBox(ErrorMCI(dwReturn, "Set parameters error: "));
        mciSendCommand(m_Device.wDeviceID, MCI_CLOSE, 0, NULL);
    }
    */
    MakeFormatList();

    UINT wReturn;
    PCMWAVEFORMAT pcmWaveFormat;
    // Set up PCMWAVEFORMAT for 11 kHz 8-bit mono.
    pcmWaveFormat.wf.wFormatTag = WAVE_FORMAT_PCM;
    pcmWaveFormat.wf.nChannels = 1;
    pcmWaveFormat.wf.nSamplesPerSec = 11025L;
    pcmWaveFormat.wf.nAvgBytesPerSec = 11025L;
    pcmWaveFormat.wf.nBlockAlign = 1;
    pcmWaveFormat.wf.nBitsPerSample = 8;
    // See if format is supported by any device in system.
    wReturn = IsFormatSupported((WAVEFORMATEX*)&pcmWaveFormat, WAVE_MAPPER);
    // Report results.
    if (wReturn == 0)
        AfxMessageBox("11 kHz 8-bit mono is supported.");
    else if (wReturn == WAVERR_BADFORMAT)
        AfxMessageBox("11 kHz 8-bit mono NOT supported.");
    else
        AfxMessageBox("Error opening waveform device.");

    return true;
}

MMRESULT SoundDialog::IsFormatSupported(LPWAVEFORMATEX pwfx, UINT uDeviceID)
{
    return (waveInOpen(
        NULL, // ptr can be NULL for query
        uDeviceID, // the device identifier
        pwfx, // defines requested format
        NULL, // no callback
        NULL, // no instance data
        WAVE_FORMAT_QUERY)); // query only, do not open device
}

bool SoundDialog::MakeFormatList()
{
    CString info;

    int m_iNumDevs=waveInGetNumDevs();
    if(m_iNumDevs==0)
    {
        AfxMessageBox("No input devices found");
        return false;
    }
    WAVEINCAPS* m_pDevCaps=new WAVEINCAPS[m_iNumDevs];
    for(int i=0; i<m_iNumDevs; i++)

```

```

{
    waveInGetDevCaps(i, &m_pDevCaps[i], sizeof(WAVEINCAPS));
    info.Format("wMid=%x, \n wPid=%x, \n szPname=%s \n dwFormats=%ld \n wChannels=%ld",
        m_pDevCaps[i].wMid, m_pDevCaps[i].wPid,
        m_pDevCaps[i].szPname, m_pDevCaps[i].dwFormats,
        m_pDevCaps[i].wChannels);
    AfxMessageBox(info);
}
return true;
}

```

/\*  
I used this call to open a MM handle for sample recording

```

WAVEFORMATEX wINEX;
WAVEIN wIN;

```

```

    wINEX.wFormatTag = WAVE_FORMAT_PCM;
    wINEX.nChannels = 2;
    wINEX.nSamplesPerSec = 16000;
    wINEX.wBitsPerSample = 8; // 8,16
    wINEX.nBlockAlign = (wINEX.wBitsPerSample * wINEX.nChannels) / 8;
    wINEX.nAvgBytesPerSec = wINEX.nSamplesPerSec * wINEX.nBlockAlign;
    wINEX.cbSize = 0;

```

```

f = waveInOpen(
    &wIN,
    WAVE_MAPPER,
    &wINEX,
    (unsigned long)waveInProc,
    0,
    CALLBACK_FUNCTION);

```

I decided to use ADPCM for a better compression, but if I select WAVE\_FORMAT\_ADPCM to wFormatTag, the waveInOpen returns an error code: 32. What's happening?

And of course, if you have a suggestion for a better compression, tell me.

Thanks

Accepted Answer

From: chensu Date: Sunday, August 02 1998 - 07:30PM PDT

You need to set the WAVEFORMATEX properly. For example,

```

// Format Tag: WAVE_FORMAT_ADPCM
// Channels: 1
// Samples Per Second: 11,025
// Avg Bytes Per Second: 5,666
// Block Alignment: 256
// Bits Per Sample: 4
// Extra Format Information: 32 bytes
// Offset Data Bytes
0xF4, 0x01, 0x07, 0x00, 0x00, 0x01, 0x00, 0x00,
0x00, 0x02, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00,
0xC0, 0x00, 0x40, 0x00, 0xF0, 0x00, 0x00, 0x00,
0xCC, 0x01, 0x30, 0xFF, 0x88, 0x01, 0x18, 0xFF

```

You may use an utility to convert a PCM wave file to an ADPCM wave file. Then, use the RiffWalk utility (it comes with the Platform SDK) to see its header information.

\*/

```

CString SoundDialog::GetWavFileName()
{
    CString filename=" ";
    CFile file;

```

```
if( !file.Open(m_FileName, File::modeRead) )
{
    AfxMessageBox("Cannot locate sound file");
}
else
{
    filename=file.GetFilePath();
    file.Close();
}
return filename;
}
```

```

#ifndef AFX_STATISTICS_H__D7670B1_C129_11D2_8051_000000000000__INCLUDED_
#define AFX_STATISTICS_H__D7670B1_C129_11D2_8051_000000000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Image/Image.h"
// Statistics.h : header file
//

////////////////////

// Statistics dialog

class Statistics : public CDialog
{
// Construction
public:
    int* st_Hist;
    long st_n;
    double st_min, st_max, st_Avg, st_StDev;
    CPoint st_HistMax;

    void Clean();
    bool Analyze(Image* pBmp, CRect* roi=NULL, int shape=0,
        double scaleX=-1.0, double scaleY=-1.0);
    CString toString();
    ~Statistics();
    Statistics(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{{AFX_DATA(Statistics)
    enum { IDD = IDD_STATISTICS_DIALOG };
    CString st_numString;
    CString st_sizeString;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Statistics)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(Statistics)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    int st_shape;
    long st_HistSum;
    double st_area, st_scaleX, st_scaleY;
    CString st_units;
    CRect st_Rect;

    void PaintHistrog(CDC* pDC, CRect r);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STATISTICS_H__D7670B1_C129_11D2_8051_000000000000__INCLUDED_)

```



```

// Statistics.cpp : implementation of the Statistics dialog
//

#include "stdafx.h"
#include "DCM.h"
#include "Statistics.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////
// Statistics dialog

Statistics::Statistics(CWnd* pParent /*=NULL*/)
: CDialog(Statistics::IDD, pParent)
{
   //{{AFX_DATA_INIT(Statistics)
    st_numString = _T("");
    st_sizeString = _T("");
    st_StDev = 0.0;
    //}}AFX_DATA_INIT
    st_Hist=0;
    Clean();
}

Statistics::~Statistics()
{
    Clean();
}

void Statistics::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Statistics)
    DDX_Text(pDX, IDC_STAT_AVG, st_Avg);
    DDX_Text(pDX, IDC_STAT_MAX, st_max);
    DDX_Text(pDX, IDC_STAT_MIN, st_min);
    DDX_Text(pDX, IDC_STAT_NUM, st_numString);
    DDX_Text(pDX, IDC_STAT_SIZE, st_sizeString);
    DDX_Text(pDX, IDC_STAT_STD, st_StDev);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(Statistics, CDialog)
    //{{AFX_MSG_MAP(Statistics)
    ON_WM_PAINT()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////
// Statistics message handlers
/*****
*
*   Assign 0 to all statistical parameters
*
*****/
void Statistics::Clean()
{
    st_shape=0; // rectangle by default
    st_Avg=0.0;
    st_max=0.0;
    st_min=0.0;
    st_n=0;
    st_StDev=0.0;
    st_scaleX=st_scaleY=-1.0;
    st_area=0.0;
    st_units="pixels";
    st_Rect=CRect(0,0,0,0);
}

```

```

    st_HistMax=CPoint(0,0);
    st_HistSum=0;
    if(st_Hist)
    {
        delete [] st_Hist;
        st_Hist=0;
    }
}

/*****
*
*   Output string for the status bar
*
*****/
CString Statistics::toString()
{
    CString cs;
    cs.Format("%ld points: [%f,%f] range, avg=%f, dev=%f",
              st_n, st_min, st_max, st_Avg, st_StDev);
    return cs;
}

/*****
*
*   Perform statistical analysis of the given region in the given image
*
*****/
bool Statistics::Analyze(Image *pBmp, CRect* roi, int shape, double scaleX, double scaleY)
{
    long x, y, dx, dy, p, x0, y0, a, b, ab;
    double temp;

    // Clean all statistical info
    Clean();
    st_shape=shape;
    if(scaleX>0.0 && scaleY>0.0)
    {
        st_scaleX=scaleX;    st_scaleY=scaleY;
        st_units="mm";
    }

    // Set image region
    if(! roi) st_Rect=CRect(0,0,pBmp->GetWidth()-1,pBmp->GetHeight()-1);
    else      st_Rect.CopyRect(roi);
    st_Rect.NormalizeRect();

    // Validate
    st_Rect.IntersectRect(st_Rect, CRect(0,0,pBmp->GetWidth()-1,pBmp->GetHeight()-1));
    st_Rect.NormalizeRect();
    if(st_Rect.IsRectEmpty()) return false;

    // Find statistics
    // 1. Average, range and number of points
    st_min=pBmp->GetPixel(st_Rect.left,st_Rect.top);    st_max=st_min;
    st_Avg=0.0; st_n=0;
    if(st_shape==1) // ellipse
    {
        x0=(st_Rect.left+st_Rect.right)/2;  a=st_Rect.right-x0;
        a *= a; if(a==0) a=1;
        y0=(st_Rect.top+st_Rect.bottom)/2;  b=st_Rect.bottom-y0;
        b *= b; if(b==0) b=1;
        ab=a*b;
    }
    for(x=st_Rect.left; x<st_Rect.right; x++)
    {
        for(y=st_Rect.top; y<st_Rect.bottom; y++)
        {
            if(st_shape==1) // ellipse
            {
                if(b*(x-x0)*(x-x0)+a*(y-y0)*(y-y0)>ab) continue;
            }
            p = pBmp->GetLuminance(x,y);
            st_Avg += p;
            if(p>st_max) st_max=p;
            else if (p<st_min) st_min=p;
        }
    }
}

```

```

        st_n ++;
    }
}
if(st_n<=0) return false;
st_Avg /= st_n;
st_area=st_n;
if(st_units=="mm") st_area *= (st_scaleX*st_scaleY);
// 2. Standard deviation and histogram
dx=st_Rect.Width()/80;    if(dx<=0) dx=1;
dy=st_Rect.Height()/80;   if(dy<=0) dy=1;
try { st_Hist=new int[(int)(st_max)+1]; }
catch(...)
{
    AfxMessageBox("Low memory, cannot allocate image histogram",
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!st_Hist)
{
    AfxMessageBox("Low memory, cannot allocate image histogram",
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
for(p=0; p<=st_max; p++) st_Hist[p]=0;

st_StDev=0.0; st_HistSum=0;
for(x=st_Rect.left; x<st_Rect.right; x += dx)
{
    for(y=st_Rect.top; y<st_Rect.bottom; y += dy)
    {
        if(st_shape==1) // ellipse
        {
            if(b*(x-x0)*(x-x0)+a*(y-y0)*(y-y0)>ab) continue;
        }
        p=pBmp->GetLuminance(x,y);
        st_Hist[p] ++;
        temp=p-st_Avg;
        st_StDev += temp*temp;
        st_HistSum ++;
    }
}
if(st_HistSum<=0) return false;
st_StDev = sqrt(st_StDev/st_HistSum);
// 3. Find histogram pick (maximum)
st_HistMax=CPoint(0,0);
for(p=0; p<st_max+1; p++)
{
    if(st_Hist[p]>st_HistMax.y)
    {
        st_HistMax.x=p;
        st_HistMax.y=st_Hist[p];
    }
}
return true;
}

/*****
*
*   Initialize dialog
*
*****/
BOOL Statistics::OnInitDialog()
{
    if(st_units=="pixels")
    {
        st_sizeString.Format("%d<x<%d, %d<y<%d pixels", st_Rect.left-1,st_Rect.right+1,
            st_Rect.top-1, st_Rect.bottom+1);
        st_numString.Format("%ld",st_n);
    }
    else /* mm */
    {
        double x0=(st_Rect.left-1)*st_scaleX;
        double x1=(st_Rect.right+1)*st_scaleX;
        double y0=(st_Rect.top-1)*st_scaleY;

```

```

double y1=(st_Rect.bottom+1)*st_scaleY;
st_sizeString.Format("%.1lf<x<%.1lf, %.1lf<y<%.1lf mm", st_n, y0, y1);
st_numString.Format("%d, Area: %.1lf mm2", st_n, st_area);
}

CDialog::OnInitDialog();

return TRUE; // return TRUE unless you set the focus to a control
// EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
* Paint dialog
*
*****/
void Statistics::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    PaintHisto(&dc, CRect(20,180,295,400));

    // Do not call CDialog::OnPaint() for painting messages
}

/*****
*
* Paint histogram in the dialog region r
*
*****/
void Statistics::PaintHisto(CDC *pDC, CRect r)
{
    if(!st_Hist) return;
    long p, pmax;
    pmax=(long)st_max+1;

    // Draw rectangle and axis
    CRect r1=r;
    r1.InflateRect(2,2);
    pDC->Rectangle(r1); // (r1,EDGE_ETCHED,BF_RECT);
    pDC->MoveTo(r.left,r.bottom); pDC->LineTo(r.right, r.bottom);
    pDC->MoveTo(r.left,r.bottom); pDC->LineTo(r.left, r.top);
    pDC->MoveTo(r.left,r.top+2); pDC->LineTo(r.left+3, r.top+2);
    pDC->MoveTo(r.left,r.bottom);

    // Set font
    pDC->SetMapMode(MM_TEXT);
    CFont *oldcf = pDC->SelectObject(&theApp.app_SmallFont);
    pDC->SetTextColor(RGB(0,0,0));
    pDC->SetBkMode(TRANSPARENT);

    // Write titles
    CString title;
    title.Format("%.2lf%%",100*(double)(st_HistMax.y)/st_HistSum);
    pDC->TextOut(r.left+5,r.top,title);
    pDC->TextOut(r.left-1,r.bottom,"0");
    if(pmax-1>st_HistMax.x+10) // max intensity
    {
        title.Format("%d",pmax-1);
        pDC->TextOut(r.right-7,r.bottom,title);
    }
    if(st_HistMax.x>5 || pmax<10) // most frequent intensity
    {
        p=r.left+((long)(st_HistMax.x)*r.Width())/pmax;
        pDC->MoveTo(p,r.bottom); pDC->LineTo(p, r.bottom-10);
        title.Format("%d",st_HistMax.x);
        pDC->TextOut(p-5,r.bottom,title);
    }

    // Plot the histogram
    pDC->MoveTo(r.left,r.bottom);
    for(p=0; p<pmax; p++)
    {
        pDC->LineTo(r.left+(p*r.Width())/pmax,

```

```
r.bottom = (long)(st_Hist[p]) * r.Height() / st_Hist[0].Max.y);
```

```
}
```

```
// Restore fonts
```

```
pDC->SelectObject(oldcf);
```

```
}
```

```

#if !defined(AFX_MACPROGRESSCTRL_H_603BBF44_B19C_11D3_90FA_0020AFBC44_499D__INCLUDED_)
#define AFX_MACPROGRESSCTRL_H_603BBF44_B19C_11D3_90FA_0020AFBC44_499D__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MacProgressCtrl.h : header file
//
// CMacProgressCtrl class, version 1.0
//
// Copyright (c) 1999 Paul M. Meidinger (pmmeidinger@yahoo.com)
//
// Feel free to modify and/or distribute this file, but
// do not remove this header.
//
// I would appreciate a notification of any bugs discovered or
// improvements that could be made.
//
// This file is provided "as is" with no expressed or implied warranty.
//
// History:
//     PMM 12/21/1999     Initial implementation.

////////////////////////////////////
// CMacProgressCtrl window

#include <afxcmn.h>
class CMacProgressCtrl : public CProgressCtrl
{
// Construction
public:
    CMacProgressCtrl();

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMacProgressCtrl)
    //}}AFX_VIRTUAL

// Implementation
public:
    BOOL GetIndeterminate();
    void SetIndeterminate(BOOL bIndeterminate = TRUE);
    COLORREF GetColor();
    void SetColor(COLORREF crColor);
    virtual ~CMacProgressCtrl();

    // Generated message map functions
protected:
    //{{AFX_MSG(CMacProgressCtrl)
    afx_msg void OnPaint();
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg BOOL OnEraseBkgnd(CDC* pDC);
    //}}AFX_MSG

    DECLARE_MESSAGE_MAP()
private:
    int m_nIndOffset;
    BOOL m_bIndeterminate;
    void DrawVerticalBar(CDC *pDC, const CRect rect);
    void DrawHorizontalBar(CDC *pDC, const CRect rect);
    void DeletePens();
    void CreatePens();
    CPen m_penColor;
    CPen m_penColorLight;
    CPen m_penColorLighter;
    CPen m_penColorDark;
    CPen m_penColorDarker;
    CPen m_penDkShadow;

```

```
CPen m_penShadow;  
CPen m_penLiteShadow;  
void GetColors();  
COLORREF m_crColor;  
COLORREF m_crColorLight;  
COLORREF m_crColorLighter;  
COLORREF m_crColorLightest;  
COLORREF m_crColorDark;  
COLORREF m_crColorDarker;  
COLORREF m_crDkShadow;  
COLORREF m_crShadow;  
COLORREF m_crLiteShadow;  
};
```

```
////////////////////////////////////
```

```
class OProgress : public CMacProgressCtrl  
{  
public:  
    void ShowProgress(int percent, char* info=NULL);  
    bool Initialize();  
    OProgress();  
    virtual ~OProgress();  
};
```

```
//{{AFX_INSERT_LOCATION}}  
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.  
#endif // !defined(AFX_MACPROGRESSCTRL_H__603BBF44_B19C_11D3_90FA_0020AFBC499D__INCLUDED_)
```

```
// MacProgressCtrl.cpp : implementation file
//
// CMacProgressCtrl class, version 1.0
//
// Copyright (c) 1999 Paul M. Meidinger (pmmeidinger@yahoo.com)
//
// Feel free to modify and/or distribute this file, but
// do not remove this header.
//
// I would appreciate a notification of any bugs discovered or
// improvements that could be made.
//
// This file is provided "as is" with no expressed or implied warranty.
//
// History:
//      PMM 12/21/1999      Initial implementation.
```

```
#include "stdafx.h"
#include "MacProgressCtrl.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
#define IDT_INDETERMINATE      100
#define IND_BAND_WIDTH        20
```

```

// Function prototypes.
COLORREF LightenColor(const COLORREF crColor, BYTE byIncreaseVal);
COLORREF DarkenColor(const COLORREF crColor, BYTE byReduceVal);

//-----
//
//
COLORREF LightenColor(const COLORREF crColor, BYTE byIncreaseVal)
//
// Return Value:      None.
//
// Parameters       :   crColor - References a COLORREF structure.
//                   byReduceVal - The amount to reduce the RGB values by.
//
// Remarks          :   Lightens a color by increasing the RGB values by the given number.
//
//
{
    BYTE byRed = GetRValue(crColor);
    BYTE byGreen = GetGValue(crColor);
    BYTE byBlue = GetBValue(crColor);

    if ((byRed + byIncreaseVal) <= 255)
        byRed = BYTE(byRed + byIncreaseVal);
    if ((byGreen + byIncreaseVal) <= 255)
        byGreen = BYTE(byGreen + byIncreaseVal);
    if ((byBlue + byIncreaseVal) <= 255)
        byBlue = BYTE(byBlue + byIncreaseVal);

    return RGB(byRed, byGreen, byBlue);
} // LightenColorref

```

```

//-----
//
COLORREF DarkenColor(const COLORREF crColor, BYTE byReduceVal)
//
// Return Value:      None.
//
// Parameters       :   crColor - References a COLORREF structure.
//                     byReduceVal - The amount to reduce the RGB values by.
//
// Remarks          :   Darkens a color by reducing the RGB values by the given number.
//
//
{
    BYTE byRed = GetRValue(crColor);
    BYTE byGreen = GetGValue(crColor);

```



```

    BYTE byBlue = GetBValue(color);

    if (byRed >= byReduceVal)
        byRed = BYTE(byRed - byReduceVal);
    if (byGreen >= byReduceVal)
        byGreen = BYTE(byGreen - byReduceVal);
    if (byBlue >= byReduceVal)
        byBlue = BYTE(byBlue - byReduceVal);

    return RGB(byRed, byGreen, byBlue);
} // DarkenColorref

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMacProgressCtrl

//-----
//
CMacProgressCtrl::CMacProgressCtrl()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    Standard constructor.
//
{
    m_bIndeterminate = FALSE;
    m_nIndOffset = 0;
    m_crColor = ::GetSysColor(COLOR_HIGHLIGHT);
    GetColors();
    CreatePens();
} // CMacProgressCtrl

//-----
CMacProgressCtrl::~CMacProgressCtrl()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    None.
//
{
    DeletePens();
} // ~CMacProgressCtrl

BEGIN_MESSAGE_MAP(CMacProgressCtrl, CProgressCtrl)
    //{AFX_MSG_MAP(CMacProgressCtrl)
    ON_WM_PAINT()
    ON_WM_TIMER()
    ON_WM_ERASEBKGD()
    //{AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CMacProgressCtrl message handlers

//-----
//
void CMacProgressCtrl::OnPaint()
//
// Return Value:    None.
//
// Parameters      :    None.
//
// Remarks         :    The framework calls this member function when Windows
//                      or an application makes a request to repaint a portion
//                      of an application's window.
//
{
    CPaintDC dcPaint(this); // device context for painting
    CRect rect, rectClient;

```

```

GetClientRect(rectClient);
rect = rectClient;
BOOL bVertical = GetStyle() & PBS_VERTICAL;

// Create a memory DC for drawing.
CDC dc;
dc.CreateCompatibleDC(&dcPaint);
int nSavedDC = dc.SaveDC();
CBitmap bmp;
bmp.CreateCompatibleBitmap(&dcPaint, rect.Width(), rect.Height());
CBitmap *pOldBmp = dc.SelectObject(&bmp);

CBrush br1(m_crColorLightest);
CBrush br2(::GetSysColor(COLOR_3DFACE));
dc.FillRect(rect, &br2);

int nLower, nUpper;
GetRange(nLower, nUpper);

// Determine the size of the bar and draw it.
if (bVertical)
{
    if (!m_bIndeterminate)
        rect.top = rect.bottom - int(((float)rect.Height() * float(GetPos() - nLower)) / float
(nUpper - nLower));
    dc.FillRect(rect, &br1);
    DrawVerticalBar(&dc, rect);
}
else
{
    if (!m_bIndeterminate)
        rect.right = int(((float)rect.Width() * float(GetPos() - nLower)) / float(nUpper - nLo
wer));
    dc.FillRect(rect, &br1);
    DrawHorizontalBar(&dc, rect);
}

dcPaint.BitBlt(rectClient.left, rectClient.top, rectClient.Width(), rectClient.Height(),
&dc, rectClient.left, rectClient.top, SRCCOPY);

dc.SelectObject(pOldBmp);
dc.RestoreDC(nSavedDC);
dc.DeleteDC();
// OnPaint
}

-----
void CMacProgressCtrl::DrawHorizontalBar(CDC *pDC, const CRect rect)
//
// Return Value:    None.
//
// Parameters      :    pDC - Specifies the device context object.
//                   rect - Specifies the rectangle of the progress bar.
//
// Remarks         :    Draws a horizontal progress bar.
//
{
    if (!rect.Width())
        return;

    int nLeft = rect.left;
    int nTop = rect.top;
    int nBottom = rect.bottom;

    // Assume we're not drawing the indeterminate state.
    CPen *pOldPen = pDC->SelectObject(&m_penColorLight);

    if (m_bIndeterminate)
    {
        pOldPen = pDC->SelectObject(&m_penColor);
        int nNumBands = (rect.Width() / IND_BAND_WIDTH) + 2;
        int nHeight = rect.Height() + 1;

        int nAdjust = nLeft - IND_BAND_WIDTH + m_nIndOffset;
    }
}

```

```

{
    // Display pattern recta
    Draw(PATTERN_RECT);

    // DoModal
    if(!AllocatePattern()) return IDCANCEL;
    CDialog::DoModal();
    DeallocatePattern();
    Erase(PATTERN_RECT);
    Erase(FOUND_RECT);
    return IDOK;
}

/*****
*
*   Prepare dialog before displaying
*
*****/
BOOL FindRegion::OnInitDialog()
{
    CDialog::OnInitDialog();
    // Force dialog to appear in the left top screen corner
    CRect wr(0,0,f_FoundDisplay.right+10,f_FoundDisplay.bottom+70);
    wr.OffsetRect(theApp.app_ScreenResolution-CSize(50,80)-wr.Size());
    MoveWindow(wr,FALSE);

    // Set parameters
    f_Correlation=-10.0;
    f_ImageStartSearchPoint=CPoint(0,0);
    f_Percent=50;
    f_CDC->SetStretchBltMode(HALFTONE);

    f_Speed.SetRange(0,5,TRUE);
    f_Speed.SetTicFreq(1);
    f_Speed.SetLineSize(2);
    f_Speed.SetPos(3);

    f_Progress.SetRange(0,100);

    return TRUE;
}

/*****
*
*   Display image pattern and found areas on the dialog
*
*****/
void FindRegion::DisplayPatterns()
{
    // Grab dialog CDC
    CDC* dialog_pDC=GetDC();
    dialog_pDC->SetStretchBltMode(HALFTONE);

    // Output and frame image patterns
    f_pBmp->DisplayDIB(dialog_pDC,f_PatternDisplay,f_ImagePatternRect,CPoint(0,0),false);
    if(f_Correlation>=-1.0)
    {
        f_pBmp->DisplayDIB(dialog_pDC,f_FoundDisplay,f_ImageFoundRect,CPoint(0,0),false);
    }
    dialog_pDC->DrawEdge(f_PatternDisplay,EDGE_RAISED,BF_RECT);
    dialog_pDC->DrawEdge(f_FoundDisplay,EDGE_RAISED,BF_RECT);

    // Select font
    dialog_pDC->SetMapMode(MM_TEXT);
    CFont *oldcf = dialog_pDC->SelectObject(&theApp.app_SmallFont);
    dialog_pDC->SetBkColor(RGB(192,192,192));

    // Draw titles
    int off = 12*theApp.app_ResolutionScaleFactor;
    dialog_pDC->TextOut(f_PatternDisplay.left, f_PatternDisplay.top-off,
        "Selected pattern : ");
    if(f_Correlation>=-1.0)
    {
        CString str_corr;
    }
}

```

```

        str_corr.Format("Match Found (%4.0f %%): ", 100*f_Correlation);
        dialog_pDC->TextOut(f_FoundDisplay.Left, f_FoundDisplay.Top, str_corr);
    }

    // Reset fonts
    dialog_pDC->SelectObject(oldcf);
}

/*****
 *
 * Draw and erase regions
 *
 *****/
void FindRegion::Draw(UINT code)
{
    CRect tmp_rect;
    switch(code)
    {
        case PATTERN_RECT:
            tmp_rect=f_ScreenPatternRect;
            tmp_rect.DeflateRect(1,1);
            f_CDC->DrawEdge(tmp_rect, EDGE_ETCHED, BF_RECT);
            tmp_rect.DeflateRect(theApp.app_ResolutionScaleFactor, theApp.app_ResolutionScaleFactor);
            f_CDC->DrawEdge(tmp_rect, EDGE_ETCHED, BF_RECT);
            break;
        case FOUND_RECT:
            tmp_rect=f_ScreenFoundRect;
            tmp_rect.DeflateRect(1,1);
            f_CDC->DrawEdge(tmp_rect, EDGE_BUMP, BF_RECT);
            break;
    }
}

void FindRegion::Erase(UINT code)
{
    switch(code)
    {
        case PATTERN_RECT:
            f_pBmp->DisplayDIB(f_CDC, f_ScreenPatternRect, f_ImagePatternRect,
                               CPoint(0,0), false);
            break;
        case FOUND_RECT:
            f_pBmp->DisplayDIB(f_CDC, f_ScreenFoundRect, f_ImageFoundRect,
                               CPoint(0,0), false);
            break;
    }
}

void FindRegion::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    DisplayPatterns();
    // Do not call CDialog::OnPaint() for painting messages
}

/*****
 *
 * Allocate and deallocate search pattern
 *
 *****/
bool FindRegion::AllocatePattern()
{
    int i,j;
    // Allocate search pattern
    int rw=f_ImagePatternRect.Width();
    int rh=f_ImagePatternRect.Height();
    if(rw<=0 || rh<=0) return false;
    try { f_Pattern=new long*[rw]; }
    catch(...) { return false; }
    if(!f_Pattern) return false;
    for(i=0; i<rw; i++)
    {
        f_Pattern[i] = new long[rh];
        if(f_Pattern[i]==0)
        {
            for(j=0; j<i; j++) { if(f_Pattern[j]) delete [] f_Pattern[j]; }
            delete [] f_Pattern;
        }
    }
}

```

```

        return false;
    } // out of memory
}

// Fill search pattern with pixel data
f_Pattern_Average=0;
int x, x0, y, y0, count=0;
for(x0=0,x=f_ImagePatternRect.left; x0<rw; x0++,x++)
{
    for(y0=0,y=f_ImagePatternRect.top; y0<rh; y0++,y++)
    {
        if(f_ReduceNoise) f_Pattern[x0][y0]=f_pBmp->GetSmoothedLuminance(x,y);
        else f_Pattern[x0][y0]=f_pBmp->GetLuminance(x,y);
        f_Pattern_Average += f_Pattern[x0][y0];
        count ++;
    }
}
if(count<1) return false;
f_Pattern_Average /= count;
return true;
}

bool FindRegion::DeallocatePattern()
{
    if(f_Pattern)
    {
        for(int i=0; i<f_ImagePatternRect.Width(); i++)
            if(f_Pattern[i]) delete [] f_Pattern[i];
        delete [] f_Pattern;
        f_Pattern=NULL;
    }
    return true;
}

```

```

#if !defined(AFX_FTPLOGINDIALOG_H__C867C413_D8A3_11D2_8070_000000000000__INCLUDED_)
#define AFX_FTPLOGINDIALOG_H__C867C413_D8A3_11D2_8070_000000000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FTPLoginDialog.h : header file
//

////////////////////
// FTPLoginDialog dialog
#include "resource.h"
class FTPLoginDialog : public CDialog
{
// Construction
public:
    void SetData(CString host, CString usr, CString pwd);
    FTPLoginDialog(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{AFX_DATA(FTPLoginDialog)
    enum { IDD = IDD_DIALOG_FTP };
    CString m_Host;
    CString m_Pwd;
    CString m_User;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(FTPLoginDialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(FTPLoginDialog)
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

//{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FTPLOGINDIALOG_H__C867C413_D8A3_11D2_8070_000000000000__INCLUDED_)

```

```
// FTPLoginDialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "FTPLoginDialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// FTPLoginDialog dialog
```

```
FTPLoginDialog::FTPLoginDialog(CWnd* pParent /*=NULL*/)
: CDialog(FTPLoginDialog::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(FTPLoginDialog)
    m_Host = _T("");
    m_Pwd = _T("");
    m_User = _T("");
    //}}AFX_DATA_INIT
}
```

```
void FTPLoginDialog::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(FTPLoginDialog)
    DDX_Text(pDX, IDC_FTP_HOST, m_Host);
    DDX_Text(pDX, IDC_FTP_PWD, m_Pwd);
    DDX_Text(pDX, IDC_FTP_USER, m_User);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(FTPLoginDialog, CDialog)
    //{{AFX_MSG_MAP(FTPLoginDialog)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// FTPLoginDialog message handlers
```

```
void FTPLoginDialog::SetData(CString host, CString usr, CString pwd)
```

```
{
    m_Host=host;    m_User=usr;    m_Pwd=pwd;
}
```

```
BOOL FTPLoginDialog::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();
    m_Pwd="";
    UpdateData(FALSE);
    return TRUE;    // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

```

#if !defined(AFX_LOGFILE_H__0B9E5823_7508_11D3_96FD_00105A21774F__INCLUDED_)
#define AFX_LOGFILE_H__0B9E5823_7508_11D3_96FD_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// LogFile.h : header file
//
#include "resource.h"

////////////////////
// LogFile dialog

class LogFile : public CDialog, public DICOMViewLog
{
// Construction
public:
    afx_msg void OnClear();
    void DoModeless(CString win_title="Client Log");
    void Load(const char *pText) { DICOMViewLog::Load(pText); }
    void Load(DICOMObject& DO) { DICOMView::Load(DO); };
    bool IsOn();
    bool RefreshText();

    LogFile(CString filename="", RTC* rtc=NULL, CWnd* pParent = NULL)
    {
        CreateDVL((char*)(LPCSTR)filename, rtc);
    };

// Dialog Data
//{{AFX_DATA(LogFile)
enum { IDD = IDD_DIALOG_LOGFILE };
CEdit m_LogWindow;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(LogFile)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(LogFile)
virtual BOOL OnInitDialog();
afx_msg void OnOK();
afx_msg void OnRefresh();
afx_msg void OnClose();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()

private:
    void OnCancel( );
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_LOGFILE_H__0B9E5823_7508_11D3_96FD_00105A21774F__INCLUDED_)

```



```

// LogFile.cpp : implementation file
//

#include "stdafx.h"
#include "DCM.h"
#include "LogFile.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////
// LogFile dialog
void LogFile::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(LogFile)
    DDX_Control(pDX, IDC_EDIT_FILE_LOG, m_LogWindow);
    DDX_Text(pDX, IDC_EDIT_FILENAME, CString(m_Filename));
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(LogFile, CDialog)
    //{{AFX_MSG_MAP(LogFile)
    ON_BN_CLICKED(IDC_BUTTON_REFRESH, OnRefresh)
    ON_BN_CLICKED(IDC_BUTTON_CLEAR, OnClear)
    ON_BN_CLICKED(IDOK, OnOK)
    ON_WM_CLOSE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////
// LogFile message handlers

/*****
*
* Set dialog window to log text
*
*****/
bool LogFile::RefreshText()
{
    if(!(this->GetSafeHwnd())) return true;    // dialog is not displayed
    long max_length=10000;
    CString tbuffer((TCHAR)(' '),max_length);

    // Refresh log text
    DicomViewLog::RefreshText((char*)(LPCSTR)tbuffer, max_length);

    // Display log
    int n=tbuffer.Replace("\n","\r\n");
    m_LogWindow.SetWindowText(tbuffer);
    UpdateData(FALSE);
    m_LogWindow.LineScroll(n+10);
    return true;
}

/*****
*
* Initialize dialog
*
*****/
BOOL LogFile::OnInitDialog()
{
    CDialog::OnInitDialog();
    RefreshText();
    return TRUE;    // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
* Display dialog
*****/

```

```

*
*****
void LogFile::DoModeless(CString win_title /*="Client Log"*/)
{
    if(this->GetSafeHwnd()) return;
    Create(IDD_DIALOG_LOGFILE,NULL);
    // Always display on top
    SetWindowPos(&wndTopMost, 0,0,0,0,
                SWP_NOMOVE | SWP_NOSIZE | SWP_SHOWWINDOW );
    SetWindowText(win_title);
    //ShowWindow(SW_SHOWNORMAL);
    if(win_title.Find("Client")>-1) // offset client window
    {
        theApp.MoveWindowToCorner(this, CSize(20,20));
    }
    else
    {
        theApp.MoveWindowToCorner(this);
    }
}

bool LogFile::IsOn() { return (this->GetSafeHwnd() != NULL); }
/*****
*
*   Buttons
*
*****/
void LogFile::OnOK() { DestroyWindow(); }
void LogFile::OnClose() { DestroyWindow(); }
void LogFile::OnCancel() { DestroyWindow(); }
void LogFile::OnRefresh() { RefreshText(); }
void LogFile::OnClear()
{
    DICOMViewLog::RefreshText(NULL, -1);
    if(this->GetSafeHwnd()) m_LogWindow.SetWindowText("");
}

```

```

#ifndef AFX_LUPA_H__24CE8B94_7D8F_11D2_958F_000000000000__INCLUDED_
#define AFX_LUPA_H__24CE8B94_7D8F_11D2_958F_000000000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// Lupa.h : header file
//

#include "Image/Image.h"
#include "resource.h"

////////////////////////////////////

// Lupa dialog

class Lupa : public CDialog
{
// Construction
public:
    Lupa(CWnd* pParent = NULL);
    ~Lupa();

// Dialog Data
    enum { IDD = IDD_MAGNIFY_DIALOG };
private:
    BOOL    m_optimize;
    long    m_height;
    long    m_width;
public:
    double m_zoom;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Lupa)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
public:
    bool    m_active;
    CSize   m_scnSize;

    void    Initialize(CSize csScn, double scrn_zoom, double lupa_zoom);
    void    Reset_DC(CDC *pDC, CRect& scrolled_client);
    void    Move(CPoint& a, CPoint& rel, Image* pBmp, CDC* pDC);
    bool    Resize(CDC* pDC, CPoint center, CPoint vertex);
    bool    Resize(CDC *pDC);
    CString toString();
    inline CRect Lupa::SetImgRect(CPoint & cpI)
    {
        CRect r( CPoint(cpI.x-(1_imgSize.cx>>1), cpI.y-(1_imgSize.cy>>1)), 1_imgSize );
        return r;
    };
    inline CRect SetScrnRect(CPoint & cpS)
    {
        return CRect(CPoint(cpS.x-(this->m_scnSize.cx>>1), cpS.y-(this->m_scnSize.cy>>1)),
            this->m_scnSize);
    };
protected:

    // Generated message map functions
    //{{AFX_MSG(Lupa)
    virtual BOOL OnInitDialog();
    afx_msg void OnChangeMagnifyWidthEdit();
    afx_msg void OnChangeMagnifyHeightEdit();
    afx_msg void OnChangeMagnifyZoomEdit();
    afx_msg void OnChangeMagnifyOptimize();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    bool    m_preactive;

```

```
double    l_img_zoom;
CSize     l_imgSize;
CRect     l_scnRect, l_dragrect;
HBITMAP   l_DIB;
BYTE*     l_Data;
CDC*       l_DC;

void       Draw(CDC *pDC);
bool       Update2(CRect& a, CRect&b);
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
line.
```

```
#endif // !defined(AFX_LUPA_H__24CE8B94_7D8F_11D2_958F_000000000000__INCLUDED_)
```

```
// Lupa.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "Lupa.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// Lupa dialog
```

```
Lupa::Lupa(CWnd* pParent /*=NULL*/)
: CDialog(Lupa::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(Lupa)
    l_height = 128*theApp.app_ResolutionScaleFactor;
    l_width = l_height;
    l_zoom = 2.0;
    l_optimize=false;
    //}}AFX_DATA_INIT
    l_active=false;
    l_preactive=false;
    l_DC=NULL;
    l_DIB=NULL;
    l_Data=NULL;
    this->Initialize(CSize(l_width,l_height), 1.0, l_zoom);
}
```

```
Lupa::~Lupa()
```

```
{
    if(l_DC && !theApp.app_Metheus) delete l_DC;
    if(l_Data) delete [] l_Data;
    if(l_DIB)
    {
        MetheusDeleteCompatibleBitmap(l_DC->GetSafeHdc(), l_DIB);
    }
}
```

```
void Lupa::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Lupa)
    DDX_Text(pDX, IDC_MAGNIFY_HEIGHT_EDIT, l_height);
    DDV_MinMaxLong(pDX, l_height, 0, 1000);
    DDX_Text(pDX, IDC_MAGNIFY_WIDTH_EDIT, l_width);
    DDV_MinMaxLong(pDX, l_width, 0, 1000);
    DDX_Text(pDX, IDC_MAGNIFY_ZOOM_EDIT, l_zoom);
    DDV_MinMaxDouble(pDX, l_zoom, 0., 10.);
    DDX_Check(pDX, IDC_MAGNIFY_OPTIMIZE, l_optimize);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(Lupa, CDialog)
```

```
    //{{AFX_MSG_MAP(Lupa)
    ON_EN_CHANGE(IDC_MAGNIFY_WIDTH_EDIT, OnChangeMagnifyWidthEdit)
    ON_EN_CHANGE(IDC_MAGNIFY_HEIGHT_EDIT, OnChangeMagnifyHeightEdit)
    ON_EN_CHANGE(IDC_MAGNIFY_ZOOM_EDIT, OnChangeMagnifyZoomEdit)
    ON_EN_CHANGE(IDC_MAGNIFY_OPTIMIZE, OnChangeMagnifyOptimize)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```

/*****
*   LUPA initialization
*
*   Input:
*   on-screen lupa size csScn, screen image zoom scrn_zoom, LUPA l_zoom l_img_zoom
*
*****/
```

```

*****
void Lupa::Initialize(CSize scnSize, double scrn_zoom, double lupa_zoom)
{
    l_scnSize=CSize(csScn.cx,csScn.cy);
    l_zoom=lupa_zoom;
    l_img_zoom=scrn_zoom;
    double ivzf=1.0 / (scrn_zoom*lupa_zoom); //combined inversed zoom
    l_imgSize=CSize((long)(l+ivzf*l_scnSize.cx),
                    (long)(l+ivzf*l_scnSize.cy));
    l_dragRect=CRect(0,0,0,0);
}

/*****
*
*   Lupa message handlers
*
*****/
BOOL Lupa::OnInitDialog()
{
    CDialog::OnInitDialog();
    l_width=l_scnSize.cx;
    l_height=l_scnSize.cy;
    UpdateData(FALSE);
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void Lupa::OnChangeMagnifyWidthEdit()
{
    UpdateData(TRUE);
    if(l_width<10) l_width=10;
    l_scnSize.cx=l_width;
    Initialize(l_scnSize, l_img_zoom, l_zoom);
}

void Lupa::OnChangeMagnifyHeightEdit()
{
    UpdateData(TRUE);
    if(l_height<10) l_height=10;
    l_scnSize.cy=l_height;
    Initialize(l_scnSize, l_img_zoom, l_zoom);
}

void Lupa::OnChangeMagnifyZoomEdit()
{
    UpdateData(TRUE);
    if(l_zoom<1.5) l_zoom=1.5;
    Initialize(l_scnSize, l_img_zoom, l_zoom);
}

void Lupa::OnChangeMagnifyOptimize()
{
    UpdateData(TRUE);
    l_optimize=!l_optimize;
}

/*****
*
*   Move lupa over the image CDC, responding to (dis)activated status
*
*****/
void Lupa::Move(CPoint& a, CPoint& rel, Image* pBmp, CDC* pDC)
{
    CPoint p;
    CRect r0,r1;
    CSize da=a-rel;

    if(l_active) // active lupa was requested
    {
        // Remove lupa resizing rectangle, if any
        if(l_dragRect.bottom != 0)
        {
            Draw(pDC);
            l_dragRect=CRect(0,0,0,0);
        }
    }
}

```

```

if(!l_preactive)    // was not active before
{
    r0=CRect(0,0,pBmp->m_ScreenMap.crScreen.Width(),pBmp->m_ScreenMap.crScreen.Height());
    Reset_l_DC(pDC,r0);
    r0=CRect(0,2,0,2); // dummy update area
}
else    r0=l_scnRect; // Remember previous image area
// Compute new lupa zoom area
r1=SetScrnRect(a);
// For Metheus: ignore rectangles not fully inside the image area
if(theApp.app_Metheus && !pBmp->m_ScreenMap.Screen_in_Image(r1,3))    return;
l_scnRect=r1;
// Compute and redraw update rectangles
bool bu=Update2(r0,r1);
if(theApp.app_Metheus)
{
    r0.InflateRect(2,2);
    pBmp->DisplayDIB(pDC,r0,pBmp->m_ScreenMap.Screen_to_Image(r0),CPoint(da));
    /*
    MetheusLoadImageFromDIB(pDC->GetSafeHdc(),l_DIB,theApp.app_DynamicPaletteStart,
        r0.left,r0.top,r0.Width(),r0.Height(),
        r0.left,r0.top);
    */
}
else
{
    r0.OffsetRect(-da);
    pDC->BitBlt(r0.left,r0.top,r0.Width(),r0.Height(),l_DC,
        r0.left,r0.top,SRCCOPY);
}
if(bu)
{
    if(theApp.app_Metheus)
    {
        r1.InflateRect(2,2);
        pBmp->DisplayDIB(pDC,r1,pBmp->m_ScreenMap.Screen_to_Image(r1),CPoint(da));
        /*
        MetheusLoadImageFromDIB(pDC->GetSafeHdc(),l_DIB,theApp.app_DynamicPaletteStart,
            r1.left,r1.top,r1.Width(),r1.Height(),
            r1.left,r1.top);
        */
    }
    else
    {
        r1.OffsetRect(-da);
        pDC->BitBlt(r1.left,r1.top,r1.Width(),r1.Height(),l_DC,
            r1.left,r1.top,SRCCOPY);
    }
}
// Zoom image into new area
r0=l_scnRect;
r0.OffsetRect(-da);
CRect ir=SetImgRect(pBmp->m_ScreenMap.Screen_to_Image(a));
if(l_optimize)
{
    Image* kadr=new Image(false); // no pixel undo
    if(!kadr->CreateImage(8*((7+ir.Width())/8),8*((7+ir.Height())/8),
        pBmp->GetBytesPerPixel(), pBmp->m_pPal))
    {
        delete kadr;
        pBmp->DisplayDIB(pDC,r0,ir,CPoint(0,0),false);
        pDC->DrawEdge(&(r0),EDGE_BUMP,BF_RECT);
        return;
    }
    kadr->m_pPal->p_active=false; // no palettes for lupa !
    pBmp->GetSubimage(kadr,ir.left,ir.top);
    kadr->TR_HistStretch(1, false);
    kadr->DisplayDIB(pDC,r0,CRect(0,0,ir.Width()-1,ir.Height()-1),CPoint(0,0),false);
    delete kadr;
}
else
{
    pBmp->DisplayDIB(pDC,r0,ir,CPoint(0,0),false);
}
}

```

```

pDC->DrawEdge(&(r0), EDGE_3D, BF_RECT);
}
else // disactivated lupa was requested
{
    if(l_preactive)
    {
        p=l_scnRect.TopLeft()-da;
        if(theApp.app_Metheus)
        {
            pBmp->DisplayDIB(pDC,l_scnRect,pBmp->m_ScreenMap.Screen_to_Image(l_scnRect),da);
            /*
            MetheusLoadImageFromDIB(pDC->GetSafeHdc(),l_DIB,theApp.app_DynamicPaletteStart,
                                   p.x,p.y,l_scnRect.Width(),l_scnRect.Height(),
                                   p.x,p.y);
            */
        }
        else
        {
            pDC->BitBlt(p.x,p.y,l_scnRect.Width(),l_scnRect.Height(),l_DC,
                      p.x,p.y,SRCCOPY);
            delete l_DC;
        }
        l_preactive=false;
        l_DC=0;
    }
}
}

```

```

/*****
Represents update region as two rectangles
Returns false if only one rectangle "a" must be updated
or true if both "a" and "b"
*****/
bool Lupa::Update2(CRect &a, CRect &b)
{
    if(a.Width()!=b.Width() || a.Height()!=b.Height() ) return false;
    if(a.EqualRect(&b)) // coinciding rectangles
    {
        a=CRect(a.left,a.top,a.left,a.top);
        return false;
    }
    CRect a1; a1.CopyRect(&a);
    CRect b1; b1.CopyRect(&b);
    if(a.left<=b.left && b.left<=a.right)
    {
        if(a.top<=b.top && b.top<=a.bottom)
        {
            a1=CRect(a.left,a.top,b.left,a.bottom);
            b1=CRect(b.left,a.top,a.right,b.top);
        }
        else if(a.top<=b.bottom && b.bottom<=a.bottom)
        {
            a1=CRect(a.left,a.top,b.left,a.bottom);
            b1=CRect(b.left,b.bottom,a.right,a.bottom);
        }
        else return false;
    }
    else if(a.left<=b.right && b.right<=a.right)
    {
        if(a.top<=b.top && b.top<=a.bottom)
        {
            a1=CRect(a.left,a.top,b.right,b.top);
            b1=CRect(b.right,a.top,a.right,a.bottom);
        }
        else if(a.top<=b.bottom && b.bottom<=a.bottom)
        {
            a1=CRect(a.left,b.bottom,b.right,a.bottom);
            b1=CRect(b.right,a.top,a.right,a.bottom);
        }
        else return false;
    }
}

```



```

    }
    else return false;
    a.CopyRect(&a1);
    b.CopyRect(&b1);
    return true;
}

/*****
*
*   Copies current screen image into lupa CDC l_DC
*
*****/
void Lupa::Reset_l_DC(CDC * pDC, CRect& scrolled_client)
{
    int w=scrolled_client.Width();
    int h=scrolled_client.Height();
    l_preactive=true;
    if(theApp.app_Metheus)
    {
        l_DC=pDC;    return;

        w=2048; h=2560;
        // BYTE buffer
        if(l_Data) { delete [] l_Data; l_Data=NULL; }
        if(l_DIB)
        {
            MetheusDeleteCompatibleBitmap(pDC->GetSafeHdc(), l_DIB);
            l_DIB=NULL;
        }
        l_DIB=MetheusCreateCompatibleBitmap(pDC->GetSafeHdc(), w, h);

        l_Data=new BYTE[w*h*3];
        HDC hdc=pDC->GetSafeHdc();
        BOOL b1=MetheusGetImageIntoData(hdc, l_DIB, theApp.app_DynamicPaletteStart,
                                         (UCHAR*)l_Data, w, h, 2*w, 16,
                                         0, 0, w, h,
                                         0, 0);

        if(b1==FALSE)
        {
            Beep(700, 150);
            AfxMessageBox("MetheusGetImageIntoData Failed");
        }
        BOOL b2=MetheusLoadImageFromData(hdc, l_DIB, theApp.app_DynamicPaletteStart,
                                         l_Data, w, h, 2*w, 16,
                                         0, 0, w, h,
                                         scrolled_client.left, scrolled_client.top);

        if(b2==FALSE)
        {
            Beep(700, 250);
            AfxMessageBox("MetheusLoadImageFromData Failed");
        }
    }
    else
    {
        if(l_DC) { l_DC->DeleteDC(); delete l_DC; l_DC=0; }
        l_DC=new CDC();
        l_DC->CreateCompatibleDC(pDC);
        CBitmap b;
        b.CreateCompatibleBitmap(pDC, w, h);
        l_DC->SelectObject(&b);
        l_DC->BitBlt(0, 0, w, h, pDC, scrolled_client.left, scrolled_client.top, SRCCOPY);
        b.DeleteObject();
    }
}

/*****
*
*   Interactively resize the Lupa region on the image
*
*****/
bool Lupa::Resize(CDC *pDC, CPoint center, CPoint vertex)
{
    Draw(pDC); // remove old rectangle

```

```

    CPoint z=vertex-center;
    int a=(abs(z.x))<<1; if(a<1) a=32;
    int b=(abs(z.y))<<1; if(b<32) b=32;
    Initialize(CSize(a,b), l_img_zoom, l_zoom);
    l_dragRect=SetScrnRect(center);
    Draw(pDC); // draw new rectangle
    return true;
}

bool Lupa::Resize(CDC *pDC)
{
    Draw(pDC); // just remove old rectangle
    l_dragRect=CRect(0,0,0,0);
    return true;
}

/*****
 *
 *   Output Lupa parameters into a string (used in status bar)
 *
 *****/
CString Lupa::toString()
{
    CString s;
    s.Format("Screen size %dx%d, zoom=%.2lf", l_scnSize.cx, l_scnSize.cy, l_zoom);
    return s;
}

/*****
 *
 *   Draw Lupa region rectangle
 *
 *****/
void Lupa::Draw(CDC *pDC)
{
    int dmode=SetROP2(pDC->m_hDC, R2_NOT);
    pDC->MoveTo(l_dragRect.TopLeft());
    pDC->LineTo(l_dragRect.right, l_dragRect.top);
    pDC->LineTo(l_dragRect.right, l_dragRect.bottom);
    pDC->LineTo(l_dragRect.left, l_dragRect.bottom);
    pDC->LineTo(l_dragRect.TopLeft());
    SetROP2(pDC->m_hDC, dmode);
}

```

```

#if !defined(AFX_RENAME_FILE_DIR_DIALOG_H__5527F676_DCE1_11D2_9627_000105A21774F__INCLUDED_)
#define AFX_RENAME_FILE_DIR_DIALOG_H__5527F676_DCE1_11D2_9627_000105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Rename_File_Dir_Dialog.h : implementation file
//

#include "stdafx.h"
#include "DCM.h"

//#ifdef _DEBUG
//#define new DEBUG_NEW
//undef THIS_FILE
//static char THIS_FILE[] = __FILE__;
//endif

////////////////////////////////////
// Rename_File_Dir_Dialog dialog

class Rename_File_Dir_Dialog : public CDialog
{
// Construction
public:
    Rename_File_Dir_Dialog(CString old_name, CWnd* pParent = NULL);    // standard constructor
    CString getName();

// Dialog Data
    //{AFX_DATA(Rename_File_Dir_Dialog)
    enum { IDD = IDD_DIALOG_RENAME_FILE_DIR };
    CString m_NewName;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(Rename_File_Dir_Dialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(Rename_File_Dir_Dialog)
    // NOTE: the ClassWizard will add member functions here
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CString m_OldName;
};

////////////////////////////////////
// Rename_File_Dir_Dialog dialog

Rename_File_Dir_Dialog::Rename_File_Dir_Dialog(CString old_name, CWnd* pParent /*=NULL*/)
: CDialog(Rename_File_Dir_Dialog::IDD, pParent)
{
    m_OldName=old_name;
    m_OldName.TrimLeft();
    m_OldName.TrimRight();
    m_OldName.Replace("\\", "/");
    //{AFX_DATA_INIT(Rename_File_Dir_Dialog)
    m_NewName = m_OldName; //_T("");
    //}AFX_DATA_INIT
}

/*****
*
*   Take care of the appropriate file/directory name syntax
*
*****/

```

```

*****
CString Rename_File_Dir_Dialog::GetName()
{
    m_NewName.Remove(' ');
    m_NewName.Remove('/');
    m_NewName.Remove('\\');
    if(m_OldName[0]=='/') m_NewName=CString("/") + m_NewName;
    if(m_OldName.Find(".dcm",-1)>-1 && m_NewName.Find(".dcm",-1)==-1)
        m_NewName += CString(".dcm");
    if(m_OldName.Find(".gz",-1)>-1 && m_NewName.Find(".gz",-1)==-1)
        m_NewName += CString(".gz");
    return m_NewName;
}

void Rename_File_Dir_Dialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Rename_File_Dir_Dialog)
    DDX_Text(pDX, IDC_EDIT_RENAME, m_NewName);
    DDV_MaxChars(pDX, m_NewName, 200);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(Rename_File_Dir_Dialog, CDialog)
    //{{AFX_MSG_MAP(Rename_File_Dir_Dialog)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// Rename_File_Dir_Dialog message handlers

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_RENAME_FILE_DIR_DIALOG_H__5527F676_DCE1_11D2_9627_00105A21774F__INCLUDED_)

```

```
// Ruler.h: interface for the Ruler class.
//
/////////////////////////////////////////////////////////////////
#ifndef AFX_RULER_H_D76C3833_C6DF_11D2_9609_00105A21774F__INCLUDED_
#define AFX_RULER_H_D76C3833_C6DF_11D2_9609_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Ruler
{
public:
    bool r_undo, r_active;

    void SetScale(int code);
    void ChangeTicksAndStyle(CDC *pDC, int d);
    void UpdatePopupMenu(CMenu* pop);
    void Redraw(CDC* pDC, CPoint& p, double* zoom=NULL, double dx=0.0, double dy=0.0);
    void Draw(CDC* pDC);
    CString toString(CPoint scroll);
    Ruler();
    virtual ~Ruler();

private:
    int r_ticks;
    double r_length, r_pix_length, r_pix_spacingX, r_pix_spacingY, r_scale_coeff;
    double* r_zoom;
    CString r_scale;
    CSize r_size;
    CPoint r_start, r_end;

    void Clean();
    void GetLength();
};

#endif // !defined(AFX_RULER_H_D76C3833_C6DF_11D2_9609_00105A21774F__INCLUDED_)
```

```
// Ruler.cpp: implementation of the Ruler class.
```

```
//
```

```
////////////////////////////////////
```

```
#include "stdafx.h"
```

```
#include "DCM.h"
```

```
#include "Ruler.h"
```

```
#ifdef _DEBUG
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
```

```
#define new DEBUG_NEW
```

```
#endif
```

```
////////////////////////////////////
```

```
// Construction/Destruction
```

```
////////////////////////////////////
```

```
Ruler::Ruler()
```

```
{
    Clean();
}
```

```
Ruler::~Ruler()
```

```
{
    Clean();
}
```

```
/*
*****
*/
```

```
* Set all to 0
```

```
*****/
```

```
void Ruler::Clean()
```

```
{
    r_active=false;
    r_undo=false;
    r_ticks=1;
    r_start=CPoint(30,30);
    r_end=CPoint(10,10);
    r_size=CSize(0,0);
    r_scale=CString("pixels");
    r_pix_spacingX=1.0; r_pix_spacingY=1.0;
    double x=1.0; r_zoom=&x;
    r_scale_coeff=1.0;
    r_length=0.0;
    r_pix_length=0.0;
}
```

```
/*
*****
*/
```

```
* Draw a ruler (once)
```

```
*****/
```

```
void Ruler::Draw(CDC *pDC)
```

```
{
    CSize t;
    int dmode=SetROP2(pDC->m_hDC, R2_NOT);
    CPen* old_pen = pDC->SelectObject(&theApp.app_Pen);
    // Draw Ruler line
    pDC->MoveTo(r_start); pDC->LineTo(r_end);
    if(r_ticks)
    {
        // Circle the start point
        pDC->Arc(CRect(r_start.x-6,r_start.y-6,r_start.x+6,r_start.y+6), r_start, r_start);
        // Circle the end point
        if(r_pix_length>16) pDC->Arc(CRect(r_end.x-6,r_end.y-6,r_end.x+6,r_end.y+6), r_end, r_end);
    }
    // Draw tick points
    if(r_pix_length>4*r_ticks)
    {
        t=CSize( (int)(0.5+4*r_size.cy/r_pix_length),

```

```

        - (0.5+4*r_size.cx/r_pix_length) ) / normal vector
CPoint tic;
for (int i=1; i<=r_ticks; i++)
{
    tic=CPoint( (i*r_start.x+(r_ticks-i+1)*r_end.x)/(r_ticks+1),
                (i*r_start.y+(r_ticks-i+1)*r_end.y)/(r_ticks+1));
    pDC->MoveTo(tic+t); pDC->LineTo(tic-t);
}
}
pDC->SelectObject(old_pen);
SetROP2(pDC->m_hDC, dmode);
}

/*****
*
*   Display ruler depending on its status
*
*****/
void Ruler::Redraw(CDC *pDC, CPoint &p, double* zoom, double dx, double dy)
{
    if(!r_active) return;
    // Set pixel spacing scales
    if(dx!=0.0)
    {
        r_pix_spacingY=r_pix_spacingX=dx;
        if(dy!=0.0) r_pix_spacingY=dy;
        if(r_pix_spacingX>0.0 && r_pix_spacingY>0.0)    r_scale="mm";
        else      r_scale="pixels";
    }

    // Choose drag point
    double ds=_hypot(r_start.x-p.x,r_start.y-p.y);
    double de=_hypot(r_end.x-p.x,r_end.y-p.y);
    if(ds<de)    // start point
    {
        if(de<5.0) return; // avoid degraded Ruler
        if(r_undo) Draw(pDC); // remove previous ruler
        r_start=CPoint(p.x,p.y);
    }
    else        // end point
    {
        if(ds<5.0) return; // avoid degraded Ruler
        if(r_undo) Draw(pDC); // remove previous ruler
        r_end=CPoint(p.x,p.y);
    }

    // Update zoom if needed
    if(zoom)
    {
        r_zoom=zoom;
        int px=(p.x<<1)-10; int py=(p.y<<1)-10;
        if(r_start.x>px) r_start.x=px; if(r_start.y>py) r_start.y=py;
        if(r_end.x>px) r_end.x=px; if(r_end.y>py) r_end.y=py;
    }

    // Draw new ruler
    GetLength();
    Draw(pDC); // new ruler
    r_undo=true;
}

/*****
*
*   Report current measurement for the status bar
*
*****/
CString Ruler::toString(CPoint scroll)
{
    CString info;
    info.Format("Distance from (%d,%d) to (%d,%d) is %.2lf ",
                r_start.x+scroll.x, r_start.y+scroll.y, r_end.x+scroll.x, r_end.y+scroll.y,
                r_length);
    return (info+r_scale);
}

```

```
// FileBrowser.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "FileBrowser.h"
#include "Rename_File_Dir_Dialog.h"
#include "CreateDirectoryDialog.h"
#include "AccurateTimer.h"
#include "Compressor.h"
#include <io.h>
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
// File list types
#define FILE 0
#define DFILE 2
#define DFILE_VU 4
#define DIR 6
#define ROOT 8
```

```

////////////////////////////////////
// FileBrowser dialog

```

```
FileBrowser::FileBrowser(CWnd* pParent /*=NULL*/)
    : CDialog(FileBrowser::IDD, pParent), m_InSession("DCM")
```

```

//{{AFX_DATA_INIT(FileBrowser)
m_Directory = CString("");
m_NumFiles = 0;
m_NumFilesTotal = 0;
m_Directory_FTP = _T("");
m_NumFiles_FTP = 0;
m_NumFilesTotal_FTP = 0;
m_SpeedInfo = _T("");
//}}AFX_DATA_INIT
m_ParentDirectory=CString("");
m_RequestedFile=CString("");
m_pFTPConnection=NULL;

```

```
FileBrowser::~FileBrowser()
```

```

    deleteFTP();
    m_InSession.Close();
    m_ImageList.DeleteImageList();
}

```

/\*\*\*\*\*

```
* Initialize main parameters
```

\*\*\*\*\*/

```
bool FileBrowser::Initialize(CString temp_dir)
```

```

if(m_ImageList.m_hImageList==NULL)
{
    m_ImageList.Create(16,17,ILC_COLOR4,9,1);
    /* Initialize Browser icons */
    HICON hicon;
    for(int nic=0; nic<9; nic++)
    {
        hicon=AfxGetApp()->LoadIcon(IDI_FILE);
        m_ImageList.Add(hicon);
    }
    hicon=AfxGetApp()->LoadIcon(IDI_DFILE);
    hicon=AfxGetApp()->LoadIcon(IDI_DFILE_VU);
    hicon=AfxGetApp()->LoadIcon(IDI_DIR);
    hicon=AfxGetApp()->LoadIcon(IDI_ROOT);
    ::DeleteObject(hicon);
    m_ImageList.Replace(DFILE,hicon);
    m_ImageList.Replace(DFILE_VU,hicon);
    m_ImageList.Replace(DIR,hicon);
    m_ImageList.Replace(ROOT,hicon);
}

```



```

}

// Get temp file name
m_TempFile=temp_dir+CString("/copy_");

// Initialize hash table of opened files
m_Opened.InitHashTable(37);

// Directory parameters
m_Directory="";
m_Directory_FTP="";

// Preset FTP parameters
m_INhost="";
m_INlogon="";
m_INpassword="";

// Display parameters
m_DCMonly_loc=true;
m_DCMonly_FTP=false;
m_FilterFTP=false;

return true;
}

void FileBrowser::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(FileBrowser)
    DDX_Control(pDX, IDC_LIST_FTP, m_List_FTP);
    DDX_Control(pDX, IDC_FILE_BROWSE_COMBO, m_DriveList);
    DDX_Control(pDX, IDC_LIST, m_List);
    DDX_Text(pDX, IDC_DIRECTORY, m_Directory);
    DDV_MaxChars(pDX, m_Directory, 200);
    DDX_Text(pDX, IDC_NUMFILES, m_NumFiles);
    DDV_MinMaxInt(pDX, m_NumFiles, -1, 1000000);
    DDX_Text(pDX, IDC_NUMFILES_TOT, m_NumFilesTotal);
    DDV_MinMaxInt(pDX, m_NumFilesTotal, -1, 100000);
    DDX_Text(pDX, IDC_DIRECTORY_FTP, m_Directory_FTP);
    DDV_MaxChars(pDX, m_Directory_FTP, 200);
    DDX_Text(pDX, IDC_NUMFILES_FTP, m_NumFiles_FTP);
    DDV_MinMaxInt(pDX, m_NumFiles_FTP, -1, 10000);
    DDX_Text(pDX, IDC_NUMFILES_TOT_FTP, m_NumFilesTotal_FTP);
    DDV_MinMaxInt(pDX, m_NumFilesTotal_FTP, -1, 10000);
    DDX_Text(pDX, IDC_HOST_FTP, m_INhost);
    DDV_MaxChars(pDX, m_INhost, 100);
    DDX_Text(pDX, IDC_SPEED_FTP, m_SpeedInfo);
    //}}AFX_DATA_MAP
}

```

```

BEGIN MESSAGE_MAP(FileBrowser, CDialog)
//{{AFX_MSG_MAP(FileBrowser)
    ON_NOTIFY(NM_DBLCLK, IDC_LIST, OnDbclkList)
    ON_CBN_SELCHANGE(IDC_FILE_BROWSE_COMBO, OnSelchangeFileBrowseCombo)
    ON_NOTIFY(LVN_COLUMNCLICK, IDC_LIST, OnColumnclickList)
    ON_BN_CLICKED(IDC_BUTTON_FTP_GET, OnButtonFtpGet)
    ON_BN_CLICKED(IDC_BUTTON_FTP_PUT, OnButtonFtpPut)
    ON_COMMAND(ID_LOCALHOST_REFRESH, OnLocRefresh)
    ON_COMMAND(ID_REMOTEHOST_REFRESH, OnFtpRefresh)
    ON_COMMAND(ID_REMOTEHOST_CONNECT, OnGoFtp)
    ON_COMMAND(ID_LOCALHOST_SHOWDICOMONLY, OnLocalhostShowDICOMonly)
    ON_UPDATE_COMMAND_UI(ID_LOCALHOST_SHOWDICOMONLY, OnUpdateLocalhostShowDICOMonly)
    ON_COMMAND(ID_REMOTEHOST_FILTERDICOMFILES, OnRemotehostFilterDICOMfiles)
    ON_UPDATE_COMMAND_UI(ID_REMOTEHOST_FILTERDICOMFILES, OnUpdateRemotehostFilterDICOMfiles)
    ON_COMMAND(ID_REMOTEHOST_SHOWDICOMONLY, OnRemotehostShowDICOMonly)
    ON_UPDATE_COMMAND_UI(ID_REMOTEHOST_SHOWDICOMONLY, OnUpdateRemotehostShowDICOMonly)
    ON_COMMAND(ID_LOCALHOST_DELETE, OnLocalhostDelete)
    ON_COMMAND(ID_REMOTEHOST_DELETE, OnRemotehostDelete)
    ON_COMMAND(ID_LOCALHOST_RENAME, OnLocalhostRename)
    ON_COMMAND(ID_REMOTEHOST_RENAME, OnRemotehostRename)
    ON_NOTIFY(NM_RCLICK, IDC_LIST, OnRclickList)
    ON_COMMAND(ID_LOCALHOST_CREATENEWDIRECTORY, OnLocalhostCreateNewDirectory)

```

```

ON_COMMAND(ID_REMOTEHOST_CREATE_NEW_DIRECTORY, OnRemotehostCreateNewDirectory)
ON_COMMAND(ID_REMOTEHOST_OPEN, OnRemotehostOpen)
ON_COMMAND(ID_LOCALHOST_OPEN, OnLocalhostOpen)
ON_UPDATE_COMMAND_UI(ID_LOCALHOST_OPEN, OnUpdateLocalhostOpen)
ON_NOTIFY(NM_RCLICK, IDC_LIST_FTP, OnRclickList)
ON_NOTIFY(NM_DBLCLK, IDC_LIST_FTP, OnDblclkList)
ON_NOTIFY(LVN_COLUMNCLICK, IDC_LIST_FTP, OnColumnclickList)
ON_COMMAND(ID_LOCALHOST_COPYTOREMOTEHOST, OnButtonFtpPut)
ON_COMMAND(ID_REMOTEHOST_COPYTOREMOTEHOST, OnButtonFtpGet)
ON_UPDATE_COMMAND_UI(ID_REMOTEHOST_OPEN, OnUpdateRemotehostOpen)
//}}AFX_MSG_MAP
ON_COMMAND(ID_BROWSER_CLOSE, CDialog::OnCancel)
ON_MESSAGE(WM_KICKIDLE, OnKickIdle)
END_MESSAGE_MAP()

////////////////////////////////////
// FileBrowser message handlers

/*****
*
*   Global-scope callback function to sort list elements
*
*****/
static int CALLBACK ListCompareFunc(LPARAM lParam1, LPARAM lParam2, LPARAM lParamSort)
{
    CString *l1=(CString*)((DWORD)lParam1);
    CString *l2=(CString*)((DWORD)lParam2);
    if(lParamSort==0)    return l1->CompareNoCase(*l2);
    else return l2->CompareNoCase(*l1);
}

/*****
*
*   Initialize DCM file browser
*
*****/
BOOL FileBrowser::OnInitDialog()
{
    CDialog::OnInitDialog();

    /* Set list icons */
    m_List_FTP.SetImageList(&m_ImageList,LVSIL_SMALL);
    m_List.SetImageList(&m_ImageList,LVSIL_SMALL);

    /* Fill the list of available drives */
    m_DriveList.ResetContent();
    m_DriveList.Dir(DDL_DRIVES | DDL_EXCLUSIVE,"");

    /* Initialize Local Browser columns */
    m_List.InsertColumn(0,NULL,LVCFMT_CENTER,18,0);
    m_List.InsertColumn(1,"Patient Name",LVCFMT_LEFT,100,1);
    m_List.InsertColumn(2,"Study Date",LVCFMT_LEFT,100);
    m_List.InsertColumn(3,"File Name",LVCFMT_LEFT,100);
    m_List.InsertColumn(4,"File Size",LVCFMT_RIGHT,100);
    m_List.InsertColumn(5,"File Creation Date",LVCFMT_RIGHT,100);
    m_List.InsertColumn(6,"Last Access Date",LVCFMT_RIGHT,100);

    /* Initialize FTP Browser columns */
    m_List_FTP.InsertColumn(0,NULL,LVCFMT_CENTER,18,0);
    m_List_FTP.InsertColumn(1,"Patient Name",LVCFMT_LEFT,100,1);
    m_List_FTP.InsertColumn(2,"Study Date",LVCFMT_LEFT,100);
    m_List_FTP.InsertColumn(3,"File Name",LVCFMT_LEFT,100);
    m_List_FTP.InsertColumn(4,"File Size",LVCFMT_RIGHT,100);
    m_List_FTP.InsertColumn(5,"File Creation Date",LVCFMT_RIGHT,100);
    m_List_FTP.InsertColumn(6,"Last Access Date",LVCFMT_RIGHT,100);

    /* Extract current local directory information */
    FindFiles(m_List,false,m_Directory);
    FindFiles(m_List_FTP,true,m_Directory_FTP);

```

```

/* enforce row selection lists */
m_List_FTP.SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE, 0, LVS_EX_FULLROWSELECT );
m_List.SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE, 0, LVS_EX_FULLROWSELECT );
// or
//ListView_SetExtendedListViewStyle(m_listControl.m_hWnd, LVS_EX_FULLROWSELECT);

```

```

/* Display updated data */
UpdateData(FALSE);

```

```

return TRUE;
}

```

```

/*****
*
*   Extract directory information
*
*   Return -1 if fails
*   or number of elements found
*
*****/

```

```

int FileBrowser::FindFiles(CListCtrl& myList, bool ftp, CString directory)
{

```

```

    BOOL bFound;
    int nitem;
    CString fname, text;
    CTime atime, ctime;
    CFileFind finder;
    CFTPFileFind* FTPfinder=NULL;

```

```

    // Get FTP connection, if needed

```

```

    if(ftp)
    {
        if(!resetFTP()) return -1;
        FTPfinder=new CFTPFileFind(m_pFTPConnection);
        if(!FTPfinder) return -1;
    }

```

```

    // Set current directory

```

```

    if(ftp)
    {
        if(directory=="") m_pFTPConnection->GetCurrentDirectory(directory);
        directory.Replace("\\", "/");
        if(m_pFTPConnection->SetCurrentDirectory(directory)==FALSE)
        {
            AfxMessageBox("Access to "+directory+CString(" denied"), MB_ICONEXCLAMATION | MB_OK);
            delete FTPfinder;
            return -1;
        }
        m_Directory_FTP=CString(directory);
    }

```

```

    else
    {

```

```

        if(directory == "")
        {

```

```

            char buffer[200];
            GetCurrentDirectory(200, buffer);
            directory=CString(buffer);
            directory.TrimRight();
        }

```

```

        directory.Replace("\\", "/");

```

```

        if(SetCurrentDirectory(directory)==FALSE)
        {

```

```

            AfxMessageBox("Cannot access directory "+directory, MB_ICONEXCLAMATION | MB_OK);
            return -1;
        }

```

```

        m_Directory=CString(directory);
    }

```

```

// OK, we can start reading files now - Clean item list

```

```

myList.DeleteAllItems();
int nfilestot=0, num_entries=0;

// Get file info
if(ftp) bFound = FTPfinder->FindFile(CString(directory)+CString("/"));
else bFound = finder.FindFile(CString(directory)+CString("/"));
BOOL isdir;
while (bFound)
{
    if(ftp)
    {
        bFound = FTPfinder->FindNextFile();
        isdir=FTPfinder->IsDirectory();
        fname=FTPfinder->GetFileName();
        text.Format("%lu",FTPfinder->GetLength());
        FTPfinder->GetLastAccessTime(ctime);
        FTPfinder->GetCreationTime(ctime);
    }
    else
    {
        bFound = finder.FindNextFile();
        isdir=finder.IsDirectory();
        fname=finder.GetFileName();
        text.Format("%lu",finder.GetLength());
        finder.GetLastAccessTime(ctime);
        finder.GetCreationTime(ctime);
    }
    fname.Replace("\\", "/");
    if(isdir) // directory
    {
        if(fname==".") continue; // do not display current directory
        if(fname=="..")
        {
            nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,ROOT,0);
            myList.SetItem(nitem,1,LVIF_TEXT,"<Parent Directory>",0,0,0,0);
            myList.SetItem(nitem,3,LVIF_TEXT,fname+"/",0,0,0,0);
        }
        else
        {
            nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,DIR,0);
            myList.SetItem(nitem,1,LVIF_TEXT,"<Subdirectory>",0,0,0,0);
            myList.SetItem(nitem,3,LVIF_TEXT,"/"+fname,0,0,0,0);
        }
        myList.SetItem(nitem,2,LVIF_TEXT,NULL,0,0,0,0);
    }
    else // file
    {
        nfilestot++;
        bool opened;
        // Process compressed or uncompressed DICOM
        if(ftp)
        {
            if(fname[0]=='.') continue; // do not process Unix system files
            opened=OpenedFilesListFind(m_Directory_FTP+CString("/") + fname);
        }
        else opened=OpenedFilesListFind(m_Directory+CString("/") + fname);

        // Grab main file data
        int filetype=FILE;
        if(opened) filetype=DFILE_VU;
        nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,filetype,0);
        myList.SetItem(nitem,1,LVIF_TEXT,"",0,0,0,0);
        myList.SetItem(nitem,2,LVIF_TEXT,"",0,0,0,0);
        myList.SetItem(nitem,3,LVIF_TEXT,fname,0,0,0,0);
    }
    myList.SetItemText(nitem,4,text);
    myList.SetItemText(nitem,5,ctime.Format("%d.%m.%Y"));
    myList.SetItemText(nitem,6,ctime.Format("%d.%m.%Y"));
    num_entries++;
}

// Get parent directory, clean up
int sl=max(directory.ReverseFind('\\'),directory.ReverseFind('/'));
if(ftp)

```

```

{
    FTPfinder->Close(); delete FTPfinder;
    if(sl<=0) m_ParentDirectory_FTP=CString(m_Directory_FTP);
    else m_ParentDirectory_FTP=CString(directory.Left(sl));
    // Set file statistics
    m_NumFilesTotal_FTP=nfilestot;
    // Set parent directory - time and size entries are invalid
    nitem=myList.InsertItem(LVIF_PARAM|LVIF_IMAGE,num_entries,NULL,0,0,ROOT,0);
    myList.SetItem(nitem,1,LVIF_TEXT,"<Parent Directory>",0,0,0,0);
    myList.SetItem(nitem,3,LVIF_TEXT,"../",0,0,0,0);
    myList.SetItem(nitem,2,LVIF_TEXT,NULL,0,0,0,0);
    myList.SetItemText(nitem,4,text); //!! invalid
    myList.SetItemText(nitem,5,ctime.Format("%d.%m.%Y")); //!! invalid
    myList.SetItemText(nitem,6,atime.Format("%d.%m.%Y")); //!! invalid
    resetFTP();
}
else
{
    finder.Close();
    if(sl<=0) m_ParentDirectory=CString(m_Directory);
    else m_ParentDirectory=CString(directory.Left(sl));
    // Set file statistics
    m_NumFilesTotal=nfilestot;
    // Set current drive
    int cl=m_Directory.Find(':');
    CString dr=CString(m_Directory.Left(cl));
    dr=CString("[-")+dr+CString("-]");
    if(m_DriveList.SelectString(-1,dr) < 0) return -1;
}

// Locate DICOM files
if(!ftp || (ftp && m_FilterFTP)) DICOM_Filter(myList, ftp);

// Sort
SortByColumn(1,myList);

// Update dialog window and exit
UpdateData(FALSE);
return m_NumFiles;
}

/*****
* Sort browser list "myList" by specified column "col"
* *****/
void FileBrowser::SortByColumn(int col, CListCtrl& myList)
{
    int entries=myList.GetItemCount();
    if(entries<2 || col<0 || myList.GetColumnWidth(col)<1) return; // Nothing to sort
    CString *keys = new CString[entries];
    if(!keys) return; // Low memory

    int itemState, col_type;
    CString ts, pname;

    // Find column type (in a block, to preserve memory)
    col_type=0;
    {
        char buffer[50];
        LVCOLUMN cl;
        cl.iOrder=col; cl.iSubItem=col; cl.pszText=buffer;
        cl.mask=LVCF_TEXT | LVCF_ORDER | LVCF_SUBITEM;
        myList.GetColumn(col,&cl);
        CString col_name=CString(buffer);
        if(col_name.Find("Date",0)>0) col_type=1; // date type
        else if(col_name.Find("Size",0)>0) col_type=2; // size type
    }

    // Set appropriate sort key
    for(int i=0; i<entries; i++)

```

```

{
    itemState=GetItemImageState(i,myList);
    // Obtain item key string
    ts=CString(myList.GetItemText(i, col));
    if(itemState==FILE || itemState==DFILE || itemState==DFILE_VU) // files
    {
        pname=CString(myList.GetItemText(i, 1));
        if(col_type==1) // date string
            keys[i]=CString(ts.Right(4)+ts.Mid(3,2)+ts.Left(2))+pname;
        else if(col_type==2) // file size
            keys[i]=CString('0',16-ts.GetLength())+ts+pname;
        else if(col!=1) // name, except patient name
            keys[i]=ts+pname;
        else keys[i]=pname+CString(myList.GetItemText(i, 3));
    }
    else keys[i]=ts; // directories

    // Account for root-directory-file priority adding priority prefix
    if(col_type==1) ts.Format("%d",itemState);
    else ts.Format("%d",9-itemState);
    keys[i] = ts+keys[i];
    myList.SetItemData(i, (DWORD) (&(keys[i])));
}
myList.SortItems(ListCompareFunc, (col_type==1));
delete [] keys;
return;
}

/*****
*
* Update state of the item "item" in the list "myList"
*
*****/
int FileBrowser::GetItemImageState(int nitem, CListCtrl& myList)
{
    LVITEM itm;
    itm.iItem=nitem;    itm.mask=LVIF_IMAGE;    // we want to retrieve only image state
    itm.iSubItem=0;
    myList.GetItem(&itm);
    return itm.iImage;
}

FileBrowser::FindName(CString fname, CListCtrl &myList)
{
    for(int n=0; n<myList.GetItemCount(); n++)
    {
        if(CString(myList.GetItemText(n,3))==fname) return n;
    }
    return -1;
}

void FileBrowser::SetItemImageState(int nitem, int state, CListCtrl& myList)
{
    LVITEM itm;
    itm.iItem=nitem;    itm.mask=LVIF_PARAM|LVIF_IMAGE; // retrieve these parameters
    m_List.GetItem(&itm);
    itm.iImage=state;    itm.iSubItem=0;
    myList.SetItem(&itm);
}

/*****
*
* Get selected position in the file list
*
*****/
int FileBrowser::GetSelectedPosition(bool local)
{
    int sel=-1;
    POSITION pos;
    if(local) pos=m_List.GetFirstSelectedItemPosition();
    else pos=m_List_FTP.GetFirstSelectedItemPosition();
    if (pos == NULL) return -1; // nothing selected

    if(local) sel = m_List.GetNextSelectedItem(pos);
}

```

```

else      sel = m_List->GetNextSelectedItem(pos);
return sel;
}

/*****
*
*   Process Double-left clicks on the files list
*
*****/
void FileBrowser::OnDbldclkList(NMHDR* pNMHDR, LRESULT* pResult)
{
    *pResult = 0;
    bool local= (pNMHDR->idFrom==IDC_LIST);
    int item=GetSelectedPosition(local);
    if(item>=0) OpenFile_or_Directory(local, item);
}

/*****
*
*   Process directory choice list
*
*****/
void FileBrowser::OnSelchangeFileBrowseCombo()
{
    CString drive;
    m_DriveList.GetLBText(m_DriveList.GetCurSel(), drive);
    drive=drive.Mid(2,drive.GetLength()-4);
    CString dcur=CString(m_Directory).Left(drive.GetLength());
    if(drive.CompareNoCase(dcur)==0) return;
    drive.MakeUpper();
    drive=drive+CString(":/");
    FindFiles(m_List, false, drive);
}

/*****
*
*   Sort column when clicked on column header
*
*****/
void FileBrowser::OnColumnclickList(NMHDR* pNMHDR, LRESULT* pResult)
{
    CListCtrl* myList;
    bool local= (pNMHDR->idFrom==IDC_LIST);
    if(local) myList=&m_List;
    else myList=&m_List_FTP;
    NM_LISTVIEW* pnm = (NM_LISTVIEW*)pNMHDR;
    int subitem=pnm->iSubItem;
    if(subitem>0) SortByColumn(subitem, (*myList));
    *pResult = 0;
}

/*****
*
*   Keeping track of opened files
*
*****/
void FileBrowser::OpenedFilesListAdd(CString fullname)
{
    m_Opened[fullname]=1;
}
bool FileBrowser::OpenedFilesListFind(CString fullname)
{
    int val=1;
    return (m_Opened.Lookup(fullname,val)==TRUE);
}
void FileBrowser::OpenedFilesListRemove(CString fullname)
{
    m_Opened.RemoveKey(fullname);
}

```

```

/*****
*
*   Start or close FTP session
*
*****/
void FileBrowser::OnGoFtp()
{
    FTPLoginDialog fld;
    fld.SetData(m_INhost, m_INlogon, m_INpassword);
    if(fld.DoModal()==IDOK)
    {
        if(!resetFTP(fld.m_Host,fld.m_User, fld.m_Pwd)) return;
        FindFiles(m_List_FTP, true);
    }
    return;
}

/*****
*
*   Functions to reset and delete FTP connection
*
*****/
bool FileBrowser::resetFTP(CString host, CString logon, CString pwd)
{
    m_INhost=host; m_INlogon=logon; m_INpassword=pwd;
    if(m_INhost=="") return false;
    try
    {
        deleteFTP();
        m_pFTPConnection=m_InSession.GetFtpConnection(host,logon,pwd);
    }
    catch(CInternetException* pEx)
    {
        TCHAR exCause[255];
        CString info;
        pEx->GetErrorMessage(exCause, 255);
        info=CString(exCause);
        info += _T("Some files may not be displayed.");
        AfxMessageBox(info, MB_ICONEXCLAMATION | MB_OK);
        deleteFTP(); pEx->Delete();
        return false;
    }
    if(!m_pFTPConnection)
    {
        AfxMessageBox("Cannot connect to remote computer",MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    return true;
}

bool FileBrowser::resetFTP()
{
    return resetFTP(m_INhost,m_INlogon,m_INpassword);
}

void FileBrowser::deleteFTP()
{
    if(m_pFTPConnection)
    {
        m_pFTPConnection->Close();
        delete m_pFTPConnection;
        m_pFTPConnection=NULL;
    }
}

/*****
*
*   Locate DICOM images in the given list
*   based on file contents
*
*****/

```



```

void FileBrowser::DICOM_Filter(CListCtrl &myList, bool ftp)
{
    int         nstate, ndcm_files=0;
    char         ctmp[64];
    CString      fname, locname, fnamepath,tmp;
    DICOMDataObject dob;

    // Filter unclassified files only
    for(int n=0; n<myList.GetItemCount(); n++)
    {
        nstate=GetItemImageState(n,myList);
        if(nstate!=FILE && nstate!=DFILE_VU) continue;
        fname=CString(myList.GetItemText(n, 3));

        // Create local file, if ftp connection
        if(ftp)
        {
            fnamepath=m_Directory_FTP+CString("/") + fname;
            locname=m_TempFile+fname;
            if(!FileTransfer(fnamepath,locname,true,
                DICOMObject::HighestPreviewFileSize)) continue;
        }
        else    locname=fnamepath=m_Directory+CString("/") + fname;

        // Uncompress if needed
        if(locname.Find(".gz",0)>0) // compressed DICOM
        {
            tmp=m_TempFile+fname+CString("_unzip");
            if(!Compressor::Z_unCompress(locname,tmp,1)) continue;
            locname=tmp;
        }

        // Load DICOM object in preview mode
        dob.Reset();
        if(!dob.LoadFromFile((char*)(LPCSTR)locname,true))
        {
            if( (!ftp && m_DCOnly_loc) || (ftp && m_DCOnly_FTP))
            {
                myList.DeleteItem(n);    n--;
            }
            continue;
        }
        DICOMRecord dfr;
        dfr.SetRecord(dob,"*");
        myList.SetItem(n,1,LVIF_TEXT,dfr.GetPatientName(),0,0,0,0);
        dfr.FormatStudyDate(ctmp,64,false);
        myList.SetItem(n,2,LVIF_TEXT,ctmp,0,0,0,0);
        if(nstate!=DFILE_VU) SetItemImageState(n,DFILE,myList);
        ndcm_files++;
        if(n%5==0)
        {
            myList.RedrawItems(1,n);    myList.UpdateWindow();
        }
    }
    if(ftp) m_NumFiles_FTP=ndcm_files;
    else    m_NumFiles=ndcm_files;
}

/*****
 *
 *   Refresh current FTP Window
 *
 *****/
void FileBrowser::OnFtpRefresh()
{
    if(!resetFTP()) return;
    FindFiles(m_List_FTP, true, m_Directory_FTP);
}

void FileBrowser::OnLocRefresh()
{
    FindFiles(m_List, false, m_Directory);
}

```

```

}

/*****
*
*   Get a binary file from remote server
*
*****/
void FileBrowser::OnButtonFtpGet()
{
    POSITION pos = m_List_FTP.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("Please select a remote file first\nby clicking on its icon",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    int n = m_List_FTP.GetNextSelectedItem(pos);
    int nstatus=GetItemImageState(n,m_List_FTP);
    if(nstatus != FILE && nstatus != DFILE && nstatus != DFILE_VU)
    {
        AfxMessageBox("Cannot copy directories",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    CString ftp_fname = CString(m_List_FTP.GetItemText(n,3));
    CString loc_fname=m_Directory+"/"+ftp_fname;
    if(FindName(ftp_fname,m_List)>0)
    {
        if(AfxMessageBox("Overwrite "+loc_fname+" ?",
            MB_ICONEXCLAMATION | MB_YESNO)==IDNO) return;
    }
    FileTransfer(m_Directory_FTP+"/"+ftp_fname,loc_fname,true,-1);
    OnLocRefresh();
    Beep(500,50);
}

/*****
*
*   Put a binary file to a remote server
*
*****/
void FileBrowser::OnButtonFtpPut()
{
    POSITION pos = m_List.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("Please select a local file first\nby clicking on its icon",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    int n = m_List.GetNextSelectedItem(pos);
    int nstatus=GetItemImageState(n,m_List);
    if(nstatus != FILE && nstatus != DFILE && nstatus != DFILE_VU)
    {
        AfxMessageBox("Cannot copy directories",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    CString loc_fname = CString(m_List.GetItemText(n,3));
    CString ftp_fname=m_Directory_FTP+"/"+loc_fname;
    if(FindName(loc_fname,m_List_FTP)>0)
    {
        if(AfxMessageBox("Overwrite "+ftp_fname+" ?",
            MB_ICONEXCLAMATION | MB_YESNO)==IDNO) return;
    }
    if(!FileTransfer(ftp_fname,m_Directory+"/"+loc_fname,false,-1))
    {
        AfxMessageBox("Cannot copy. \nMake sure you have remote host connected",MB_ICONEXCLAMATION | MB_OK);
        return;
    }
    OnFtpRefresh();
    Beep(400,50);
}

```

```

/*****
*
*   Handle all kinds of file transfers
*   between local and remote computers
*
*****/
bool FileBrowser::FileTransfer(CString ftp_name, CString loc_name, bool to_local,
                               int size)
{
    static bool failed=false;
    BOOL done;
    double vremia;
    if(size==0) return true;    // empty file
    CInternetFile* ifile=NULL;
    _iobuf* floc;
    BYTE* bf=NULL;

    // Refresh FTP connection
    if(!resetFTP()) return false;

    // Try to copy
    try
    {
        if(size<0) // Transfer the entire file at once
        {
            long fsize=0;
            AccurateTimer ac;   ac.Begin();
            if(to_local) done=m_pFTPConnection->GetFile(ftp_name,loc_name,FALSE);
            else done=m_pFTPConnection->PutFile(loc_name, ftp_name);
            // Find size of the copied file
            {
                _iobuf* t_file;
                t_file=fopen(loc_name,"rb");
                if (t_file != NULL)
                {
                    fsize = _filelength(_fileno(t_file)); // local file size
                    fclose(t_file);
                }
            }
            vremia=ac.End();
            if(vremia>0.1 && fsize>0)
            {
                m_SpeedInfo.Format(" %.3lf  Kbytes/sec",fsize/(1000.0*vremia));
                UpdateData(FALSE);
            }
            return (done==TRUE);
        }
        else if(size>0)
        {
            try { bf=new BYTE[size]; }
            catch(...) { return false; }
            if(!bf) return false;
            // Open local and remote files
            if(to_local)
            {
                ifile=m_pFTPConnection->OpenFile(ftp_name,GENERIC_READ,
                                                  FTP_TRANSFER_TYPE_BINARY,1);
                floc=fopen(loc_name,"wb");
            }
            else
            {
                ifile=m_pFTPConnection->OpenFile(ftp_name,GENERIC_WRITE,
                                                  FTP_TRANSFER_TYPE_BINARY,1);
                floc=fopen(loc_name,"rb");
            }
            bool can_transfer = (floc!=NULL && ifile!=NULL);
            if(can_transfer)
            {
                int lu;
                if(to_local)
                {
                    lu=ifile->Read(bf,size);
                    fwrite(bf,1,lu,floc);
                }
            }
        }
    }
}

```

```

        else
        {
            lu=fread(bf,1,size,floc);
            ifile->Write(bf,lu);
        }
        delete [] bf;    bf=NULL;
        fclose(floc);    ifile->Close(); delete ifile;
        return can_transfer;
    }
}
catch(CInternetException* ex)
{
    if(!failed)
    {
        ex->ReportError(MB_ICONEXCLAMATION | MB_OK);
        failed=true;
    }
    ex->Delete();
    if(bf) delete [] bf;
    if(ifile)
    {
        try { ifile->Close(); } catch (CInternetException* ex2) { ex2->Delete();}
        delete ifile;
    }
    return false;
}
return false;
}

/*****
*
* Local: "Show DICOMs only" message handler
*
*****/
void FileBrowser::OnLocalhostShowDICOMonly()
{
    m_DCMonly_loc =! m_DCMonly_loc;
    OnLocRefresh();
    Beep(500,50);
}

void FileBrowser::OnUpdateLocalhostShowDICOMonly(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_DCMonly_loc);
}

/*****
*
* Remote: "Filter DICOM" message handler
*
*****/
void FileBrowser::OnRemotehostFilterDICOMfiles()
{
    m_FilterFTP = !m_FilterFTP;
    OnFtpRefresh();
    Beep(400,50);
}

void FileBrowser::OnUpdateRemotehostFilterDICOMfiles(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_FilterFTP);
}

/*****
*
* Remote: "Show DICOM only" message handler
*
*****/
void FileBrowser::OnRemotehostShowDICOMonly()
{
    if(!m_FilterFTP) return;
    m_DCMonly_FTP =! m_DCMonly_FTP;
    OnFtpRefresh();
    Beep(400,50);
}

```

```

}
void FileBrowser::OnUpdateRemoteHostShowDICOMOnly(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_FilterFTP);
    pCmdUI->SetCheck(m_DCOMonly_FTP && m_FilterFTP);
}

/*****
*
*   Rename selected file/directory on the current computer
*   (local or FTP)
*
*****/
bool FileBrowser::RenameFile_or_Directory(CListCtrl &myList, bool is_local)
{
    POSITION pos = myList.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("No item selected",MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    int n = myList.GetNextSelectedItem(pos);
    int nstatus=GetItemImageState(n,myList);
    bool isfile=(nstatus == FILE || nstatus == DFILE || nstatus == DFILE_VU);
    if(!is_local && !isfile)
    {
        AfxMessageBox("Cannot rename remote directories",MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    BOOL done;
    CString name =CString(myList.GetItemText(n,3));
    Rename_File_Dir_Dialog rfdd(name);
    if(rfdd.DoModal() != IDCANCEL) return false;
    if(is_local)
    {
        if(isfile) done=MoveFile(m_Directory+"/"+name,m_Directory+"/"+rfdd.getName());
        else done=MoveFile(m_Directory+name,m_Directory+rfdd.getName());
    }
    else
    {
        if(!resetFTP()) return false;
        if(isfile) done=m_pFTPConnection->Rename(m_Directory_FTP+"/"+name,
                                                m_Directory_FTP+"/"+rfdd.m_NewName);
    }
    if(done==FALSE)
    {
        AfxMessageBox("Cannot rename "+name,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    else
    {
        if(is_local) OnLocRefresh();
        else OnFtpRefresh();
    }
    return true;
}

/*****
*
*   Create a new directory (local or ftp)
*
*****/
bool FileBrowser::CreateNewDirectory(CListCtrl &myList, bool is_local)
{
    CreateDirectoryDialog cdd;

```

```

if(cdd.DoModal()==IDCANCEL) return false;
CString name=cdd.GetName();
if(name.IsEmpty())
{
    AfxMessageBox("Cannot create: empty name",MB_ICONEXCLAMATION | MB_OK);
    return false;
}
if(FindName(name, myList)>-1)
{
    AfxMessageBox("Cannot create: already exists",MB_ICONEXCLAMATION | MB_OK);
    return false;
}
BOOL done;
if(is_local)    done=CreateDirectory(m_Directory+name, NULL);
else
{
    if(!resetFTP()) return false;
    done=m_pFTPConnection->CreateDirectory(m_Directory_FTP+name);
}
if(done==FALSE)
{
    AfxMessageBox("Cannot create "+name,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
else
{
    if(is_local)    OnLocRefresh();
    else            OnFtpRefresh();
}
return true;
}

void FileBrowser::OnLocalhostCreateNewDirectory()
{
    CreateNewDirectory(m_List, true);
    Beep(500,50);
}

void FileBrowser::OnRemotehostCreateNewDirectory()
{
    CreateNewDirectory(m_List_FTP, false);
    Beep(400,50);
}

/*****
*
*   Open selected file/directory from the current computer
*   (local or FTP)
*
*****/
void FileBrowser::OnLocalhostOpen()
{
    POSITION pos = m_List.GetFirstSelectedItemPosition();
    if (pos == NULL)    return; // nothing selected

    int item = m_List.GetNextSelectedItem(pos); //single selection
    OpenFile_or_Directory(true, item);
}

void FileBrowser::OnUpdateLocalhostOpen(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_List.GetFirstSelectedItemPosition()!=NULL);
}

```

```

void FileBrowser::OnRemoteHostOpen()
{
    POSITION pos = m_List_FTP.GetFirstSelectedItemPosition();
    if (pos == NULL) return; // nothing selected

    int item = m_List_FTP.GetNextSelectedItem(pos); //single selection
    OpenFile_or_Directory(false, item);
}

void FileBrowser::OnUpdateRemoteHostOpen(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(m_List_FTP.GetFirstSelectedItemPosition() != NULL);
}

void FileBrowser::OpenFile_or_Directory(bool local, int item)
{
    if(item < 0) return; // something is wrong
    CListCtrl* myList;
    if(local) myList = &m_List;
    else myList = &m_List_FTP;

    // Update file browser
    m_RequestedFile = CString("/") + CString(myList->GetItemText(item, 3));
    switch(GetItemImageState(item, (*myList)))
    {
        case FILE:
        case DFILE_VU:
        case DFILE:
            SetItemImageState(item, DFILE_VU, (*myList));
            if(local)
            {
                m_RequestedFile = CString(m_Directory) + m_RequestedFile;
                OpenedFilesListAdd(m_RequestedFile);
            }
            else
            {
                CString tm = CString(m_Directory_FTP) + m_RequestedFile;
                OpenedFilesListAdd(tm);
                m_RequestedFile = m_TempFile + CString(myList->GetItemText(item, 3));
                if(!FileTransfer(tm, m_RequestedFile, true, -1)) return; // cannot download
            }
            CDialog::OnOK();
            break; // unchecked file
        case DIR:
            if(local) FindFiles((*myList), false, CString(CString(m_Directory) + CString(myList->GetItemText(item, 3))));
            else FindFiles((*myList), true, CString(CString(m_Directory_FTP) + CString(myList->GetItemText(item, 3))));
            return; // switch to subdirectory
        case ROOT:
            if(local) FindFiles((*myList), false, m_ParentDirectory);
            else FindFiles((*myList), true, m_ParentDirectory_FTP);
            return; // switch to parent directory
        default: return; // do nothing
    }
    myList->Update(item);
}

/*****
*
* Delete selected file/directory from the current computer
* (local or FTP)
*
*****/
bool FileBrowser::DeleteFile_or_Directory(CListCtrl &myList, bool is_local)
{
    POSITION pos = myList.GetFirstSelectedItemPosition();
    if (pos == NULL)
    {
        AfxMessageBox("No item selected", MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    int n = myList.GetNextSelectedItem(pos);
    int nstatus = GetItemImageState(n, myList);
    bool isfile = (nstatus == FILE || nstatus == DFILE || nstatus == DFILE_VU);
    BOOL done;

```

```

CString name =CString(myList.GetItemText(n,3));
if( AfxMessageBox("Do you really want to delete "+name+" ?"
    MB_ICONEXCLAMATION | MB_YESNO) == IDNO ) return false;
if(is_local)
{
    if(isfile)        done=DeleteFile(m_Directory+"/"+name);
    else              done=RemoveDirectory(m_Directory+name);
}
else
{
    if(!resetFTP()) return false;
    if(isfile)        done=m_pFTPConnection->Remove(m_Directory_FTP+"/"+name);
    else              done=m_pFTPConnection->RemoveDirectory(m_Directory_FTP+name);
}
if(done==FALSE)
{
    AfxMessageBox("Cannot delete "+name,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
else
{
    if(is_local)      OnLocRefresh();
    else              OnFtpRefresh();
}
return true;
}

```

```

void FileBrowser::OnLocalhostRename()
{
    RenameFile_or_Directory(m_List, true);
    Beep(500,50);
}

```

```

void FileBrowser::OnRemotehostRename()
{
    RenameFile_or_Directory(m_List_FTP, false);
    Beep(400,50);
}

```

```

/*
 * Local: "Delete" message handler
 */

```

```

void FileBrowser::OnLocalhostDelete()
{
    DeleteFile_or_Directory(m_List, true);
    Beep(500,50);
}

```

```

/*
 * Remote: "Delete" message handler
 */

```

```

void FileBrowser::OnRemotehostDelete()
{
    DeleteFile_or_Directory(m_List_FTP, false);
    Beep(400,50);
}

```

```

/*
 * Support for OnUpdate messages in the menu
 */

```

```

LRESULT FileBrowser::OnKickIdle(WPARAM, LPARAM)
{
    CMenu* pMainMenu = GetMenu();
    if(!pMainMenu) return FALSE;
    CCmdUI cmdUI;

```



```

for (UINT n = 0; n < pMainMenu->GetMenuItemCount(); ++n)
{
    CMenu* pSubMenu = pMainMenu->GetSubMenu(n);
    if(!pSubMenu) continue;
    cmdUI.m_nIndexMax = pSubMenu->GetMenuItemCount();
    for (UINT i = 0; i < cmdUI.m_nIndexMax; ++i)
    {
        cmdUI.m_nIndex = i;
        cmdUI.m_nID = pSubMenu->GetMenuItemID(i);
        cmdUI.m_pMenu = pSubMenu;
        cmdUI.DoUpdate(this, FALSE);
    }
}
return TRUE;
}

```

```

/*****
 *
 *   Dynamic popup, context-sensitive menu
 *   for browser list items
 *
 *****/
void FileBrowser::OnRcClickList(NMHDR* pNMHDR, LRESULT* pResult)
{
    // Grab necessary parameters
    CListCtrl* myList;
    bool local= (pNMHDR->idFrom==IDC_LIST);
    if(local) myList=&m_List;   else myList=&m_List_FTP;
    // Click positioning
    CPoint p(GetMessagePos());
    myList->ScreenToClient(&p);
    int item=myList->HitTest(CPoint(2,p.y));    if(item<0) return; // something is wrong

    // Create popup menu
    CMenu menu;
    menu.CreatePopupMenu();

    CMenu menu1;
    menu1.LoadMenu(IDR_MENU_FILE_BROWSE);

    CMenu* pop=menu1.GetSubMenu(local?1:2);

    // Set acceptable menu IDs
    UINT ids[5]={    ID_LOCALHOST_OPEN,
                     ID_LOCALHOST_COPYTOREMOTEHOST,
                     ID_LOCALHOST_DELETE,
                     ID_LOCALHOST_RENAME,
                     ID_LOCALHOST_CREATENEWDIRECTORY};

    if(!local)
    {
        ids[0]=ID_REMOTEHOST_OPEN;
        ids[1]=ID_REMOTEHOST_COPYTOREMOTEHOST;
        ids[2]=ID_REMOTEHOST_DELETE;
        ids[3]=ID_REMOTEHOST_RENAME;
        ids[4]=ID_REMOTEHOST_CREATENEWDIRECTORY;
    }

    // Create the popup menu
    int n=0;
    unsigned int nid;
    CString strMenu;
    for(unsigned int i=0; i<pop->GetMenuItemCount(); i++)
    {
        nid=pop->GetMenuItemID(i);
        if( nid!=ids[0] && nid!=ids[1] && nid!=ids[2]
           && nid!=ids[3] && nid!=ids[4]) continue;
        pop->GetMenuString(i, strMenu, MF_BYPOSITION);
        int status=pop->GetMenuState(i, MF_BYPOSITION);
        menu.InsertMenu(n, status, nid, strMenu);
        n++;
    }
}

```

```
// Display popup menu
ClientToScreen(&p);
p=CPoint(p.x+(local?25:440),p.y+25);
menu.TrackPopupMenu(TPM_LEFTALIGN, p.x,p.y,this);
pop->DestroyMenu();
menu.DestroyMenu();
menu1.DestroyMenu();
```

1. The first part of the paper is devoted to a review of the literature on the topic. It starts with a general overview of the field, followed by a more detailed discussion of the specific issues at hand. The author then presents his own findings, which are based on a series of experiments. These findings are then compared with the results of previous studies, and the author discusses the implications of his work. Finally, the paper concludes with a summary of the main points and some suggestions for future research.

```

#if !defined(AFX_FINDREGION_2307E526_1F4B_11D3_96A4_00105A21774F__INCLUDED_)
#define AFX_FINDREGION_H_2307E526_1F4B_11D3_96A4_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define CANNOT_COMPARE -1313

#include "Image/Image.h"
// FindRegion.h : header file
//

/////////////////////////////////////////////////////////////////
// FindRegion dialog

class FindRegion : public CDialog
{
// Construction
public:
    bool Initialize(CDC *pDC, Image *pBmp, CRect &ScreenPatternRect, CSize& scroll);
    FindRegion(CWnd* pParent = NULL); // standard constructor
    ~FindRegion();

// Dialog Data
    //{AFX_DATA(FindRegion)
    enum { IDD = IDD_DIALOG_FIND_SIMILAR };
    int f_Percent;
    BOOL f_ReduceNoise;
    CProgressCtrl f_Progress;
    CSliderCtrl f_Speed;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(FindRegion)
    public:
        virtual int DoModal();
    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(FindRegion)
    afx_msg void OnButtonFindBest();
    afx_msg void OnPaint();
    afx_msg void OnButtonFindNext();
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    long f_Pattern_Average;
    long** f_Pattern;
    double f_Correlation;
    CPoint f_ImageStartSearchPoint;
    CSize f_scroll;
    CRect f_ImageFoundRect, f_ScreenFoundRect;
    CRect f_ImagePatternRect, f_ScreenPatternRect;
    CRect f_FoundDisplay, f_PatternDisplay;
    Image* f_pBmp;
    CDC* f_CDC;

    void Draw(UINT code);
    void Erase(UINT code);
    void DisplayPatterns();
    bool AllocatePattern();
    bool DeallocatePattern();
    bool Find(UINT mode);
};

```



```
// FindRegion.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "FindRegion.h"
```

```
#define PATTERN_RECT    0
#define FOUND_RECT      1
#define FIND_BEST_REGION    0
#define FIND_NEXT_REGION   1
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// FindRegion dialog
```

```
FindRegion::FindRegion(CWnd* pParent /*=NULL*/)
: CDialog(FindRegion::IDD, pParent)
{
    // Some very basic initialization
    int n=theApp.app_ResolutionScaleFactor;
    f_PatternDisplay=CRect( CPoint(175,40), CSize(100*n,100*n) );
    f_FoundDisplay= f_PatternDisplay+CPoint(0,f_PatternDisplay.bottom+10*n);
    f_Pattern=NULL;
    f_Pattern_Average=0;

    //{AFX_DATA_INIT(FindRegion)
    f_Percent = 50;
    f_ReduceNoise = TRUE;
    //}AFX_DATA_INIT
}
```

```
FindRegion::~FindRegion() { }
```

```
void FindRegion::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(FindRegion)
    DDX_Text(pDX, IDC_DIALOGFIND_PERCENT, f_Percent);
    DDV_MinMaxInt(pDX, f_Percent, 0, 100);
    DDX_Control(pDX, IDC_DIALOGFIND_PROGRESS, f_Progress);
    DDX_Control(pDX, IDC_DIALOGFIND_SLIDER, f_Speed);
    DDX_Check(pDX, IDC_DIALOGFIND_DENoise, f_ReduceNoise);
    //}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(FindRegion, CDialog)
    //{AFX_MSG_MAP(FindRegion)
    ON_BN_CLICKED(IDC_DIALOGFIND_FIND, OnButtonFindBest)
    ON_WM_PAINT()
    ON_BN_CLICKED(IDC_DIALOGFIND_NEXT, OnButtonFindNext)
    //}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// FindRegion message handlers
```

```
*****
```

```
*
*   Initialize find parameters
*
```

```
*****/
```

```
bool FindRegion::Initialize(CDC *pDC, Image *pBmp, CRect &ScreenPatternRect, CSize& scroll)
{
    f_pBmp=pBmp;    if(!f_pBmp) return false;
    f_CDC=pDC;      if(!f_CDC)  return false;
    f_scroll=scroll;
    f_ScreenPatternRect=f_ScreenFoundRect=ScreenPatternRect;
}
```

```

f_ImagePatternRect=f_ImageFoundRect
=f_pBmp->m_ScreenMap.Screen_to_Image(f_ScreenFoundRect, f_scroll);

// Force selected region inside the image, if necessary
bool inside=true;
if(f_ImagePatternRect.left<0)
{
    f_ImagePatternRect.left=0;    inside=false;
}
if(f_ImagePatternRect.top<0)
{
    f_ImagePatternRect.top=0;    inside=false;
}
if(f_ImagePatternRect.right>=f_pBmp->GetWidth())
{
    f_ImagePatternRect.right=f_pBmp->GetWidth()-1;    inside=false;
}
if(f_ImagePatternRect.bottom>=f_pBmp->GetHeight())
{
    f_ImagePatternRect.bottom=f_pBmp->GetHeight()-1;    inside=false;
}
if(!inside)
{
    f_ImagePatternRect.NormalizeRect();
    if(f_ImagePatternRect.IsRectEmpty())    return FALSE;
    f_ImageFoundRect=f_ImagePatternRect;
    f_ScreenPatternRect=f_ScreenFoundRect=
        f_pBmp->m_ScreenMap.Image_to_Screen(f_ImageFoundRect, f_scroll);
}
return true;
}

*****
* Find best match
*
*****/
bool FindRegion::Find(UINT mode)
{
    int x,y,dx,dy;
    double temp_corr;
    CPoint best_fit(0,0);
    int rw=f_ImagePatternRect.Width();
    int rh=f_ImagePatternRect.Height();
    if(f_pBmp->GetWidth()-rw<rw || f_pBmp->GetHeight()-rh<rh)
    {
        AfxMessageBox("Search pattern is too large\n"
            "Please cancel search and select smaller pattern",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    // Set coordinate increments
    int accur=f_Speed.GetPos();
    if(accur==f_Speed.GetRangeMax())
    {
        dx=dy=1;    // highest accuracy
    }
    else
    {
        dx=__max(1,rw>>accur);    dy=__max(1,rh>>accur);
    }

    // Perform fit search
    bool found_something=false;
    for(x=f_ImageStartSearchPoint.x; x<f_pBmp->GetWidth()-rw; x += dx)
    {
        if(x%10==0) f_Progress.SetPos(100*x/(f_pBmp->GetWidth()-rw));
        if(abs(x-f_ImagePatternRect.left)<rw)    continue;
        for(y=f_ImageStartSearchPoint.y; y<f_pBmp->GetHeight()-rh; y += dy)
        {
            if(abs(y-f_ImagePatternRect.top)<rh)    continue;
            temp_corr=f_pBmp->Compare(COMPARE_CORRELATION, CPoint(x,y), f_Pattern,
                rw, rh, f_Pattern_Average);
            if(temp_corr==CANNOT_COMPARE)    continue;

```

```

temp_corr = (1.0 - temp_corr)/2; // map into [0,1]
if(temp_corr > f_Correlation) // best current
{
    found_something=true;
    f_Correlation=temp_corr;
    best_fit=CPoint(x,y);
    if(mode==FIND_NEXT_REGION)
    {
        f_ImageStartSearchPoint=CPoint(x,y+1);
        goto found;
    }
    temp_corr=f_pBmp->Compare(COMPARE_CORRELATION, CPoint(x,y), f_Pattern,
                             rw, rh, f_Pattern_Average);
}
if(f_ImageStartSearchPoint.y>0) f_ImageStartSearchPoint.y=0;
} // next y
if(f_ImageStartSearchPoint.x>0) f_ImageStartSearchPoint.x=0;
} // next x

if(mode==FIND_NEXT_REGION)
{
    AfxMessageBox("Finished searching the image\n"
                  "Next search will start from the beginning.",
                  MB_ICONINFORMATION);
    f_ImageStartSearchPoint=CPoint(0,0);
    return false;
}
if(!found_something)
{
    AfxMessageBox("Not found");
    return false;
}

found:
    f_Progress.SetPos(0);
    if(best_fit.x==CANNOT_COMPARE && best_fit.y==0) return false;
    Beep(500,100);

    // Display the result
    if(f_ImageFoundRect.EqualRect(f_ImagePatternRect)==FALSE) Erase(FOUND_RECT); // remove previous
    f_ImageFoundRect=CRect(best_fit,f_ImagePatternRect.Size());
    f_ScreenFoundRect=f_pBmp->m_ScreenMap.Image_to_Screen(f_ImageFoundRect,f_scroll);
    Draw(FOUND_RECT);
    return true;
}

/*****
*
* Find Button click
*
*****/
void FindRegion::OnButtonFindBest()
{
    f_ImageStartSearchPoint=CPoint(0,0);
    f_Correlation=0.0;
    if(!Find(FIND_BEST_REGION)) return;
    DisplayPatterns();
    f_ImageStartSearchPoint=CPoint(0,0);
}

void FindRegion::OnButtonFindNext()
{
    UpdateData(TRUE);
    f_Correlation=f_Percent/100.0;
    if(!Find(FIND_NEXT_REGION)) return;
    DisplayPatterns();
    //f_ImageStartSearchPoint=f_ImageFoundRect.TopLeft()+CSize(1,1);
}

/*****
*
* Display modal dialog, and erase all rectangles after
*
*****/
int FindRegion::DoModal()

```

```
// CustomFileDialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "CustomFileDialog.h"
#include <dlgs.h>
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CCustomFileDialog
```

```
IMPLEMENT_DYNAMIC(CCustomFileDialog, CFileDialog)
```

```
BEGIN_MESSAGE_MAP(CCustomFileDialog, CFileDialog)
//{{AFX_MSG_MAP(CCustomFileDialog)
ON_BN_CLICKED(IDC_SELECT_ITEMS, OnSelectButton)
ON_WM_CONTEXTMENU()
//}}AFX_MSG_MAP
ON_COMMAND(ID_HELP, OnHelp)
END_MESSAGE_MAP()
```

```
// Filter string
CString CCustomFileDialog::szCustomDefFilter(_T("All Files (*.*)|*.*|Directories|.||"));
CString CCustomFileDialog::szCustomDefExt(_T("dcm"));
CString CCustomFileDialog::szCustomDefFileName(_T("This is the initial default file name"));
CString CCustomFileDialog::szCustomTitle(_T("Select File or Directory"));
```

```
CCustomFileDialog::CCustomFileDialog(BOOL bOpenFileDialog, DWORD dwFlags,
    LPCTSTR lpszFilter, // = szCustomDefFilter
    LPCTSTR lpszDefExt, // = szCustomDefExt
    LPCTSTR lpszFileName, // = szCustomDefFileName
    CWnd* pParentWnd) : // = NULL
    CFileDialog(bOpenFileDialog, lpszDefExt, lpszFileName, dwFlags, lpszFilter, pParentWnd)
```

```
{
    m_bMulti = FALSE;
    m_SelectSubdirectories = TRUE;

    // Most of the "customization" of CCustomFileDialog is set by the dwFlags
    // passed in to the constructor. Attempts to enable these features after
    // constructing the CCustomFileDialog, such as accessing the m_ofn structure
    // directly, may cause CCustomFileDialog to not work correctly
```

```
    m_szBigBuffer[0]='\0';
    m_ofn.lpstrFile = m_szBigBuffer;
```

```
    if (dwFlags & OFN_ALLOWMULTISELECT)
    {
        m_bMulti = TRUE;
        // MFC only provides a 260 character buffer for lpstrFile
        // This is not sufficient when you may be expecting a large number of files.
        m_ofn.nMaxFile = sizeof(m_szBigBuffer);
        if (lpszFileName != NULL)
            lstrcpyn(m_szBigBuffer, lpszFileName, sizeof(m_szBigBuffer));
    }
```

```
    else m_ofn.nMaxFile = _MAX_PATH;
```

```
    if (dwFlags & OFN_EXPLORER)
    {
        if (dwFlags & OFN_ENABLETEMPLATE)
        {
            // give it a custom title, too.
            SetTitle("Select File or Directory");
        }
    }
```

```
    else
    {
        if (m_ofn.Flags & OFN_EXPLORER)
        {
            // MFC added it, but we don't want it
            m_ofn.Flags &= ~(OFN_EXPLORER | OFN_SHOWHELP);
        }
    }
```



```

    }

// FILEOPENORD & MULTIFILEOPENORD can be found from the Infoviewer
// at Samples -> MFC Samples -> General MFC Samples -> CLIPART -> COMMDDL.GRC
// These are the customized versions
if (m_bMulti)
    SetTemplate(CUSTOM_MULTIFILEOPENORD, IDD_CUSTOM_FILE_DIALOG);
else
    SetTemplate(CUSTOM_FILEOPENORD, IDD_CUSTOM_FILE_DIALOG);
}

void CCustomFileDialog::SetTitle(CString title)
{
    CCustomFileDialog::szCustomTitle = title;
    m_ofn.lpstrTitle = CCustomFileDialog::szCustomTitle;
}

void CCustomFileDialog::DoDataExchange(CDataExchange* pDX)
{
    CFileDialog::DoDataExchange(pDX);
    ///{AFX_DATA_MAP(CCustomFileDialog)
    DDX_Check(pDX, IDC_CHECK, m_SelectSubdirectories);
    ///}AFX_DATA_MAP
}

BOOL CCustomFileDialog::ReadListViewNames()
{
    // Okay, this is the big hack of the sample, I admit it.
    // With some creative use of the Spy++ utility, you will
    // find that the listview is not actually ID = lst1 as
    // documented in some references, but is actually a child
    // of dlg item ID = lst2

    // WARNING! Although this is a non-intrusive customization,
    // it does rely on unpublished (but easily obtainable)
    // information. The Windows common file dialog box implementation
    // may be subject to change in future versions of the
    // operating systems, and may even be modified by updates of
    // future Microsoft applications. This code could break in such
    // a case. It is intended to be a demonstration of one way of
    // extending the standard functionality of the common dialog boxes.

    CWnd* pWnd = GetParent()->GetDlgItem(lst2);
    if (pWnd == NULL) return FALSE;

    CListCtrl* wndLst1 = (CListCtrl*) (pWnd->GetDlgItem(1));

    UINT nSelected = wndLst1->GetSelectedCount();
    if (!nSelected) return FALSE; // nothing selected

    CString strDirectory = GetFolderPath();
    if (strDirectory.Right(1) != _T("\\"))
    {
        strDirectory += _T("\\");
    }

    CString strItemText;
    // Could this iteration code be cleaner?
    for (int nItem = wndLst1->GetNextItem(-1, LVNI_SELECTED);
        nSelected-- > 0; nItem = wndLst1->GetNextItem(nItem, LVNI_SELECTED))
    {
        strItemText = strDirectory + wndLst1->GetItemText(nItem, 0);
        m_listDisplayNames.AddHead(strItemText);
    }
    return TRUE;
}

BOOL CCustomFileDialog::OnFileNameOK()
{
    ReadListViewNames();
    // CFileDialog's m_ofn.lpstrFile will contain last set of selections!
    return FALSE;
}

```

```

}

void CCustomFileDialog::OnSelectButton()
{
    ReadListViewNames();
    ((CDialog*)GetParent())->EndDialog(IDOK);
}

void CCustomFileDialog::OnContextMenu(CWnd* pWnd, CPoint point)
{
    const DWORD helpIDs[] =
    {
        //IDC_SELECT_ITEMS,      IDC_SELECT_ITEMS + 0x50000,
        0,0
    };
    ::WinHelp(pWnd->m_hWnd, AfxGetApp()->m_pszHelpFilePath,
        HELP_CONTEXTMENU, (DWORD)(LPVOID)helpIDs);
}

void CCustomFileDialog::OnHelp()
{
    // TODO: How do I bring up the main topic as a contextpopup?
    AfxGetApp()->WinHelp(1, HELP_CONTEXTPOPUP);
}

BOOL CCustomFileDialog::OnInitDialog()
{
    CFileDialog::OnInitDialog();

    // Clear all old selections
    m_listDisplayNames.RemoveAll();

    // Let's change some button text
    // Here's one of the big differences from pre MFC 4.0
    // customizations. All of the controls on the Explorer
    // dialog are now on a dialog which is the PARENT of
    // the CFileDialog. That's right; you'll need to do a
    // GetParent() first, before using GetDlgItem().

    // Overall, this removes the edit control and its static
    // text and then moves the filetypes combo and static
    // text up into the same spot.
    CRect rcWndEdit, rcWndStatic;

    CWnd* pWnd = GetParent()->GetDlgItem(edt1);
    pWnd->GetWindowRect(&rcWndEdit);
    GetParent()->ScreenToClient(&rcWndEdit);
    pWnd = GetParent()->GetDlgItem(stc3);
    pWnd->GetWindowRect(&rcWndStatic);
    GetParent()->ScreenToClient(&rcWndStatic);
    // More undocumented but non-implementation functions
    // Work only when supplying a customized template
    HideControl(edt1); // Hide edit control
    HideControl(stc3); // Hide edit control's text

    pWnd = GetParent()->GetDlgItem(cmb1);
    pWnd->SetWindowPos(NULL, rcWndEdit.left, rcWndEdit.top, 0, 0,
        SWP_NOSIZE | SWP_NOZORDER | SWP_NOACTIVATE);
    pWnd = GetParent()->GetDlgItem(stc2);
    pWnd->SetWindowPos(NULL, rcWndStatic.left, rcWndStatic.top, 0, 0,
        SWP_NOSIZE | SWP_NOZORDER | SWP_NOACTIVATE);

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
 *
 * Return selection # index
 *
 *****/
CString CCustomFileDialog::GetSelectedAt(UINT index)
{
    CString res("");

```

```
if(index >= (UINT)m_listDisplayNames.GetCount()) return res;
POSITION pos = m_listDisplayNames.FindIndex(index);
if(!pos) return res;
res = (CString)m_listDisplayNames.GetAt(pos);
return res;
```

```

#if !defined(AFX_COLORDIALOG_H__13C30C65_8287_11D2_9596_000000000000__INCLUDED_)
#define AFX_COLORDIALOG_H__13C30C65_8287_11D2_9596_000000000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ColorDialog.h : header file
//

////////////////////
// ColorDialog dialog

class ColorDialog : public CDialog
{
// Construction
public:
    ColorDialog(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(ColorDialog)
    enum { IDD = IDD_DIALOG_COLOR };
    CSliderCtrl m_Contrast;
    CSliderCtrl m_Gamma;
    CSliderCtrl m_Brightness;
    int         m_br_value;
    double      m_gamma_value;
    int         m_con_value;
    }//}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(ColorDialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(ColorDialog)
    virtual BOOL OnInitDialog();
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    }//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    CWnd* m_pView;

    //{{AFX_INSERT_LOCATION}}
    // Microsoft Developer Studio will insert additional declarations immediately before the previous
    // line.

#endif // !defined(AFX_COLORDIALOG_H__13C30C65_8287_11D2_9596_000000000000__INCLUDED_)

```

```
// ColorDialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "ColorDialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// ColorDialog dialog
```

```
ColorDialog::ColorDialog( CWnd* pParent /*=NULL*/)
: CDialog(ColorDialog::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(ColorDialog)
    m_br_value = 0;
    m_gamma_value = 100;
    m_con_value = 0;
    //}}AFX_DATA_INIT
    m_pView=pParent;
}
```

```
void ColorDialog::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(ColorDialog)
    DDX_Control(pDX, IDC_SLIDER_CONTRAST, m_Contrast);
    DDX_Control(pDX, IDC_SLIDER_GAMMA, m_Gamma);
    DDX_Control(pDX, IDC_SLIDER_BRIGHTNESS, m_Brightness);
    DDX_Text(pDX, IDC_SLIDER_BR_NUMBER, m_br_value);
    DDV_MinMaxInt(pDX, m_br_value, -100, 100);
    DDX_Text(pDX, IDC_SLIDER_GAM_NUMBER, m_gamma_value);
    DDV_MinMaxDouble(pDX, m_gamma_value, 0, 600);
    DDX_Text(pDX, IDC_SLIDER_CT_NUMBER, m_con_value);
    DDV_MinMaxInt(pDX, m_con_value, 0, 101);
    //}}AFX_DATA_MAP
    if (pDX->m_bSaveAndValidate)
    {
        m_br_value=m_Brightness.GetPos();
        m_gamma_value=m_Gamma.GetPos();
        m_con_value=m_Contrast.GetPos();
    }
    else
    {
        m_Brightness.SetPos(m_br_value);
        m_Gamma.SetPos((int)m_gamma_value);
        m_Contrast.SetPos(m_con_value);
    }
}
```

```
BEGIN_MESSAGE_MAP(ColorDialog, CDialog)
    //{{AFX_MSG_MAP(ColorDialog)
    ON_WM_HSCROLL()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// ColorDialog message handlers
```

```
BOOL ColorDialog::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();
    m_Brightness.SetRange(-100,100,TRUE);
    m_Brightness.SetTicFreq(20);
    m_Brightness.SetLineSize(1);
    m_Brightness.SetPageSize(10);
}
```

```

m_Brightness.SetPos(m_br_value);

m_Gamma.SetRange(0,600,TRUE);
m_Gamma.SetTicFreq(100);
m_Gamma.SetLineSize(1);
m_Gamma.SetPageSize(10);
m_Gamma.SetPos((int)m_gamma_value);
CString value;
value.Format("%3.1f",m_gamma_value/100.0);
GetDlgItem(IDC_SLIDER_GAM_NUMBER)->SetWindowText(value);

m_Contrast.SetRange(0,100,TRUE);
m_Contrast.SetTicFreq(10);
m_Contrast.SetLineSize(1);
m_Contrast.SetPageSize(10);
m_Contrast.SetPos(m_con_value);

return TRUE; // return TRUE unless you set the focus to a control
              // EXCEPTION: OCX Property Pages should return FALSE
}

void ColorDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    CString value;
    int nControl = pScrollBar->GetDlgCtrlID();
    CSliderCtrl* pControl = (CSliderCtrl*) GetDlgItem(nControl);

    switch (nControl)
    {
        case IDC_SLIDER_BRIGHTNESS:
            ASSERT(pControl != NULL);
            m_br_value = pControl->GetPos();
            value.Format("%d",m_br_value);
            GetDlgItem(IDC_SLIDER_BR_NUMBER)->SetWindowText(value);
            break;

        case IDC_SLIDER_CONTRAST:
            ASSERT(pControl != NULL);
            m_con_value = pControl->GetPos();
            value.Format("%d",m_con_value);
            GetDlgItem(IDC_SLIDER_CT_NUMBER)->SetWindowText(value);
            break;

        case IDC_SLIDER_GAMMA:
            ASSERT(pControl != NULL);
            m_gamma_value = pControl->GetPos();
            value.Format("%3.1f",m_gamma_value/100.0);
            GetDlgItem(IDC_SLIDER_GAM_NUMBER)->SetWindowText(value);
            break;

        default:
            CDialo:~g::OnHScroll(nSBCode, nPos, pScrollBar);
            break;
    }
    /* For Proof later
    if (m_pView)
    {
        m_pView->Invalidate();
        m_pView->UpdateWindow();
    }
    */

    return;
}

```

```
// Compressor.h: interface for the Compressor class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_COMPRESSOR_H__200F3A11_CF87_11D2_9617_00105A21774F__INCLUDED_
#define AFX_COMPRESSOR_H__200F3A11_CF87_11D2_9617_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "MainFrm.h"

// adding zlib stuff
#ifndef _WINDOWS
#define _WINDOWS
#endif
#ifndef ZLIB_DLL
#define ZLIB_DLL
#endif
#include "Zlib113\\zlib.h"
#define BUFLen 32768

class Compressor
{
public:
    Compressor();
    virtual ~Compressor();
    static bool Z_Compress(CString infile, CString outfile);
    static bool Z_unCompress(CString infile, CString outfile, int max_steps=0);
}

#endif // !defined(AFX_COMPRESSOR_H__200F3A11_CF87_11D2_9617_00105A21774F__INCLUDED_)
```

```
// Compressor.cpp: implementation of the Compressor class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DCM.h"
#include "Compressor.h"
#include <io.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

Compressor::Compressor()
{

}

Compressor::~Compressor()
{

}

/*****
*
* Uncompress GZipped file
* If "max_steps">0, uncompresses only first max_steps*BUFLen bytes
*
*****/
bool Compressor::Z_unCompress(CString infile, CString outfile, int max_steps)
{
    long in_size;
    CString message;
    FILE *out;
    gzFile in;
    bool gz_format_error=false;

    // Find compressed file size
    {
        FILE* t_in;
        t_in=fopen(infile,"rb");
        if (t_in == NULL)
        {
            message.Format("Cannot read from file %s",infile);
            if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
            return false;
        }
        in_size = _filelength(_fileno(t_in)); // input file size
        fclose(t_in);
    }

    // Open files for reading/writing
    in=gzopen(infile,"rb");
    if (in == NULL)
    {
        message.Format("Cannot read from file %s",infile);
        if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }

    out = fopen(outfile, "wb");
    if (out == NULL)
    {
        message.Format("Cannot write to file %s",outfile);
        if(max_steps>0) AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
}
```



```

// Put progress control in the main frame status bar
theApp.ShowProgress(1, "Compressing file with gzip");

// Perform decompression
static char buf[BUFLen];
long len, tot_len=0, steps=0;
for (;;)
{
    if(max_steps==0) theApp.ShowProgress(min(100, (33*tot_len)/in_size)); // assume 3:1 compress
ion ratio
    len = gzread(in, buf, BUFLen);
    tot_len += len;
    if (len <= 0) break; // eof
    if ((long)fwrite(buf, 1, len, out) != len && !gz_format_error)
    {
        if(max_steps>0) AfxMessageBox("Possible gz format error", MB_ICONEXCLAMATION | MB_OK);
        gz_format_error=true;
    }
    steps++;
    if(max_steps>0 && steps>=max_steps) break;
}

// Clean up
if (fclose(out))
{
    message.Format("Cannot close file %s", outfile);
    if(max_steps>0) AfxMessageBox(message, MB_ICONEXCLAMATION | MB_OK);
    return false;
}
if (gzclose(in) != Z_OK)
{
    //message.Format("Cannot close file %s", infile);
    //if(max_steps>0) AfxMessageBox(message, MB_ICONEXCLAMATION | MB_OK);
    //return false;
}
return true;
}

/*****
*
* Compress GZipped file
*
*****/
bool Compressor::Z_Compress(CString infile, CString outfile)
{
    long in_size;
    CString message;
    FILE *in;
    gzFile out;
    bool gz_format_error=false;

    // Open files for reading/writing
    in=fopen(infile, "rb");
    if (in == NULL)
    {
        message.Format("Cannot read from file %s", infile);
        AfxMessageBox(message, MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    in_size = _filelength(_fileno(in)); // input file size
    out = gzopen(outfile, "wb");
    if (out == NULL)
    {
        message.Format("Cannot write to file %s", outfile);
        AfxMessageBox(message, MB_ICONEXCLAMATION | MB_OK);
        return false;
    }

    // Put progress control in the main frame status bar
    theApp.ShowProgress(1, "Compressing file with gzip");

```

```

// Perform compression
static char buf[BUFLen];
long len, tot_len=0;
for (;;)
{
    theApp.ShowProgress((100*tot_len)/in_size);
    len=(long)fread(buf, 1, BUFLen, in);
    tot_len += len;
    if (len <= 0) break; // eof
    if(gzwrite(out, buf, len) != len && !gz_format_error)
    {
        AfxMessageBox("Possible gz format error",MB_ICONEXCLAMATION | MB_OK);
        gz_format_error=true;
    }
}

// Clean up
theApp.ShowProgress(0);
if (gzclose(out) != Z_OK)
{
    message.Format("Cannot close file %s", outfile);
    AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
if (fclose(in))
{
    message.Format("Cannot close file %s", infile);
    AfxMessageBox(message,MB_ICONEXCLAMATION | MB_OK);
    return false;
}
return true;

```

```

#if !defined(AFX_CREATEDIRECTORYDIALOG_H__EE93B006_DE6F_11D2_9629_0000105A21774F__INCLUDED_)
#define AFX_CREATEDIRECTORYDIALOG_H__EE93B006_DE6F_11D2_9629_0000105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// CreateDirectoryDialog.h : implementation file
//

#include "stdafx.h"
#include "DCM.h"

//#ifdef _DEBUG
//#define new DEBUG_NEW
//undef THIS_FILE
//static char THIS_FILE[] = __FILE__;
//endif

////////////////////////////////////
// CreateDirectoryDialog dialog

class CreateDirectoryDialog : public CDialog
{
// Construction
public:
    CreateDirectoryDialog(CWnd* pParent = NULL);    // standard constructor
    CString GetName();

// Dialog Data
   //{{AFX_DATA(CreateDirectoryDialog)
    enum { IDD = IDD_DIALOG_CREATE_DIRECTORY };
    CString m_DirName;
    }{{AFX_DATA}

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CreateDirectoryDialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    }{{AFX_VIRTUAL}

// Implementation
protected:

    // Generated message map functions
   //{{AFX_MSG(CreateDirectoryDialog)
    // NOTE: the ClassWizard will add member functions here
    }{{AFX_MSG}
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////
// CreateDirectoryDialog dialog

CreateDirectoryDialog::CreateDirectoryDialog(CWnd* pParent /*=NULL*/)
    : CDialog(CreateDirectoryDialog::IDD, pParent)
{
    //{{AFX_DATA_INIT(CreateDirectoryDialog)
    m_DirName = _T("");
    }{{AFX_DATA_INIT}
}

void CreateDirectoryDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CreateDirectoryDialog)
    DDX_Text(pDX, IDC_EDIT_DIRECTORY_CREATE, m_DirName);
    }{{AFX_DATA_MAP}
}

```

```

/*****
 *
 *   Take care of the appropriate file/directory name syntax
 *
 *****/
CString CreateDirectoryDialog::getName()
{
    m_DirName.Remove(' ');
    m_DirName.Remove('/');
    m_DirName.Remove('\\');
    m_DirName = CString("/") + m_DirName;
    return m_DirName;
}

BEGIN_MESSAGE_MAP(CreateDirectoryDialog, CDialog)
    //{{AFX_MSG_MAP(CreateDirectoryDialog)
    // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////
// CreateDirectoryDialog message handlers

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_CREATEDIRECTORYDIALOG_H__EE93B006_DE6F_11D2_9629_00105A21774F__INCLUDED_)

```

```

#if !defined(AFX_EDGESDIALOG_H__03D704B4_834D_11D2_9597_00000000__INCLUDED_)
#define AFX_EDGESDIALOG_H__03D704B4_834D_11D2_9597_00000000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// EdgesDialog.h : header file
//

////////////////////
// EdgesDialog dialog

class EdgesDialog : public CDialog
{
// Construction
public:
    BYTE GetThreshold();
    EdgesDialog(int threshold=128,CWnd* pParent = NULL);    // standard constructor

// Dialog Data
   //{{AFX_DATA(EdgesDialog)
    enum { IDD = IDD_DIALOG_EDGES };
    CSliderCtrl m_SliderEdges;
    BYTE        m_Thr_Value;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(EdgesDialog)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(EdgesDialog)
    virtual BOOL OnInitDialog();
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
}

//{{AFX_INSERT_LOCATION}}
//Microsoft Developer Studio will insert additional declarations immediately before the previous
line.

#endif // !defined(AFX_EDGESDIALOG_H__03D704B4_834D_11D2_9597_00000000__INCLUDED_)

```

```
// EdgesDialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "EdgesDialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// EdgesDialog dialog
```

```
EdgesDialog::EdgesDialog(int threshold /*=128*/, CWnd* pParent /*=NULL*/)
: CDialog(EdgesDialog::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(EdgesDialog)
    //}}AFX_DATA_INIT
    m_Thr_Value=(100*threshold)/256;
}
```

```
void EdgesDialog::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(EdgesDialog)
    DDX_Control(pDX, IDC_SLIDER_EDGES, m_SliderEdges);
    DDX_Text(pDX, IDC_SLIDER_EG_NUMBER, m_Thr_Value);
    //DDV_MinMaxByte(pDX, m_Thr_Value, 0, 100);
    //}}AFX_DATA_MAP
    if (pDX->m_bSaveAndValidate)
    {
        m_Thr_Value=m_SliderEdges.GetPos();
    }
    else
    {
        m_SliderEdges.SetPos(m_Thr_Value);
    }
}
```

```
BEGIN_MESSAGE_MAP(EdgesDialog, CDialog)
```

```
    //{{AFX_MSG_MAP(EdgesDialog)
    ON_WM_HSCROLL()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// EdgesDialog message handlers
```

```
BOOL EdgesDialog::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();
    m_SliderEdges.SetRange(0,100,TRUE);
    m_SliderEdges.SetTicFreq(10);
    m_SliderEdges.SetLineSize(1);
    m_SliderEdges.SetPageSize(10);
    m_SliderEdges.SetPos(m_Thr_Value);
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
void EdgesDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
```

```
{
    CString value;
    int nControl = pScrollBar->GetDlgCtrlID();
    CSliderCtrl* pControl = (CSliderCtrl*) GetDlgItem(nControl);

    switch (nControl)
    {
```



```
// Email.h: interface for the Email class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_EMAIL_H_FDCA3A87_F053_11D3_9789_00105A21774F__INCLUDED_
#define AFX_EMAIL_H_FDCA3A87_F053_11D3_9789_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <mapi.h>

class Email
{
public:
    bool Send(CString to, CString subject, CString text,
              CString fileattachment, HWND hWnd=NULL);
    Email();
    virtual ~Email();

protected:
    HINSTANCE      m_hlibMAPI;
    LHANDLE        m_lhSession;
    LPMAPISENDMAIL m_MAPISendMail;
    LPMAPILOGON    m_MAPILogon;
    LPMAPILOGOFF   m_MAPILogOff;
    HWND           m_hWndParent;

private:
    bool LogOff();
    bool LogOn(HWND hWnd=NULL);
};

#endif // !defined(AFX_EMAIL_H_FDCA3A87_F053_11D3_9789_00105A21774F__INCLUDED_)
```



```

// Email.cpp: implementation of the Email class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Email.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Email::Email()
{
    m_hWndParent=0;
    m_lhSession=0;
    m_hlibMAPI=0;
    m_MAPILogon = 0;
    m_MAPILogOff = 0;
    m_MAPISendMail = 0;
}

Email::~Email()
{
}

/* *****
 * Log to the Simple MAPI service
 * ***** */
bool Email::LogOn(HWND hWnd /*=NULL*/)
{
    if(m_lhSession!=0)
    {
        AfxMessageBox("Cannot load email service");
        return false;
    }
    // Load Simple MAPI dll
    m_hlibMAPI = LoadLibrary("MAPI32.DLL"); // 32 bit clients
    if(!m_hlibMAPI)
    {
        AfxMessageBox("Cannot load email service");
        return false;
    }
    m_hWndParent = hWnd;

    // Load MAPI functions
    m_MAPILogon = (LPMAPILOGON)GetProcAddress(m_hlibMAPI, "MAPILogon");
    m_MAPILogOff = (LPMAPILOGOFF)GetProcAddress(m_hlibMAPI, "MAPILogoff");
    m_MAPISendMail = (LPMAPISENDMAIL)GetProcAddress(m_hlibMAPI, "MAPISendMail");

    // Logon to MAPI
    switch(m_MAPILogon((ULONG)m_hWndParent, NULL, NULL,
        MAPI_PASSWORD_UI/* | MAPI_FORCE_DOWNLOAD*/, 0L, &m_lhSession))
    {
    case MAPI_E_INSUFFICIENT_MEMORY:
        AfxMessageBox("Insufficient memory to proceed");
        return false;
    case MAPI_E_LOGIN_FAILURE:
        AfxMessageBox("Failed to log on");
        return false;
    case MAPI_E_TOO_MANY_SESSIONS:
        AfxMessageBox("Failed: Too many email sessions open simultaneously");
        return false;
    case MAPI_E_USER_ABORT:
        AfxMessageBox("Failed: User abort");
        return false;
    }
}

```

```

    case SUCCESS_SUCCESS:
        return true;
    default: // MAPI_E_FAILURE:
        AfxMessageBox("Failed to start email service");
        return false;
    }
    return false; // must never get here
}

/*****
 *
 *   Log off the Simple MAPI service,
 *   clean MAPI memory buffers (if needed)
 *
 *****/
bool Email::LogOff()
{
    if(m_MAPILogOff(m_lhSession, (ULONG)m_hWndParent, 0L, 0L) != SUCCESS_SUCCESS)
    {
        AfxMessageBox("Failed to terminate email session");
        return false;
    }
    m_lhSession=0;
    return true;
}

/*****
 *
 *   Send the message
 *
 *****/
bool Email::Send(CString to, CString subject, CString text,
                CString fileattachment, HWND hWnd/*=NULL*/)
{
    if(!LogOn(hWnd)) return false;
    const ULONG ulReserved = 0L;

    // Initialize file attachment structure
    MapiFileDesc attachment;
    CString name=fileattachment;
    if(fileattachment!="")
    {
        int n = fileattachment.ReverseFind('/');
        if(n<0) n = fileattachment.ReverseFind('\\');
        if(n>0) name=fileattachment.Mid(n+1);
        attachment.ulReserved=ulReserved;
        attachment.flFlags=0;
        attachment.nPosition=(ULONG)-1;
        attachment.lpszPathName=(char*)(LPCSTR)fileattachment;
        attachment.lpszFileName=(char*)(LPCSTR)name;
        attachment.lpFileType=NULL;
    }

    // Initialize "to" recipient
    CString toAddress=CString("SMTP:")+to;
    MapiRecipDesc recips;
    recips.ulReserved = ulReserved;
    recips.ulRecipClass = MAPI_TO;
    recips.lpszName = (char*)(LPCSTR)to;
    recips.lpszAddress = (char*)(LPCSTR)toAddress;
    recips.ulEIDSize = 0;
    recips.lpEntryID = NULL;

    // Initialize MAPI message for one recipient, with at most one attachment
    MapiMessage message;
    message.ulReserved = ulReserved;
    message.lpszSubject = (char*)(LPCSTR)subject;
    message.lpszNoteText = (char*)(LPCSTR)text;
    message.lpszMessageType = NULL;
    message.lpszDateReceived = NULL;
    message.lpszConversationID = NULL;
    message.flFlags = 0;
    message.lpOriginator = NULL;
    message.nRecipCount = 1;

```

```

message.lpRecips = &recip;
if(fileattachment!="")
{
    message.nFileCount = 1;
    message.lpFiles = &attachment;
}
else
{
    message.nFileCount = 0;
    message.lpFiles = NULL;
}
ULONG err=m_MAPISendMail ( m_lhSession, (ULONG)m_hWndParent,
                           &message, MAPI_DIALOG,0L);

switch(err)
{
case MAPI_E_AMBIGUOUS_RECIPIENT:
    AfxMessageBox("Send failed:\nAmbiguous recipient description");
    break;
case MAPI_E_ATTACHMENT_NOT_FOUND:
    AfxMessageBox("Send failed:\nThe specified attachment was not found");
    break;
case MAPI_E_ATTACHMENT_OPEN_FAILURE:
    AfxMessageBox("Send failed:\nThe specified attachment could not be opened");
    break;
case MAPI_E_BAD_RECIPTYPE:
    AfxMessageBox("Send failed:\nBad recipient type");
    break;
case MAPI_E_INSUFFICIENT_MEMORY:
    AfxMessageBox("Send failed:\nInsufficient memory");
    break;
case MAPI_E_INVALID_RECIPS:
    AfxMessageBox("Send failed:\nInvalid recipient(s)");
    break;
case MAPI_E_LOGIN_FAILURE:
    AfxMessageBox("Send failed:\nFailed to log on successfully");
    break;
case MAPI_E_TEXT_TOO_LARGE:
    AfxMessageBox("Send failed:\nThe text in the message was too large");
    break;
case MAPI_E_TOO_MANY_FILES:
    AfxMessageBox("Send failed:\nThere were too many file attachments");
    break;
case MAPI_E_TOO_MANY_RECIPIENTS:
    AfxMessageBox("Send failed:\nThere were too many recipients");
    break;
case MAPI_E_UNKNOWN_RECIPIENT:
    AfxMessageBox("Send failed:\nUnknown recipient");
    break;
case MAPI_E_USER_ABORT:
    AfxMessageBox("Send failed:\nUser abort");
    break;
case SUCCESS_SUCCESS:
    break;
default: // MAPI_E_FAILURE:
    AfxMessageBox("Send failed");
    break;
}
return (err==SUCCESS_SUCCESS) && LogOff();
}

```

G

[illegible]

```

#if !defined(AFX_FILEBROWSER_H_B5F74D80_CC60_11D2_9614_00105A21774F__INCLUDED_)
#define AFX_FILEBROWSER_H_B5F74D80_CC60_11D2_9614_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
//FileBrowser.h : header file
//

#include "afxinet.h"
#include "FTPLoginDialog.h" // Added by ClassView

////////////////////////////////////
// FileBrowser dialog

class FileBrowser : public CDialog
{
// Construction
public:
    bool Initialize(CString temp_dir);
    CString m_RequestedFile;
    FileBrowser(CWnd* pParent = NULL); // standard constructor
    ~FileBrowser(); // destructor

// Dialog Data
//{{AFX_DATA(FileBrowser)
enum { IDD = IDD_DIALOG_FILE_BROWSE };
CListCtrl m_List_FTP;
CComboBox m_DriveList;
CListCtrl m_List;
CString m_Directory;
int m_NumFiles;
int m_NumFilesTotal;
CString m_Directory_FTP;
int m_NumFiles_FTP;
int m_NumFilesTotal_FTP;
CString m_INhost;
CString m_SpeedInfo;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(FileBrowser)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
//{{AFX_MSG(FileBrowser)
virtual BOOL OnInitDialog();
afx_msg void OnDblclkList(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnSelchangeFileBrowseCombo();
afx_msg void OnColumnclickList(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnButtonFtpGet();
afx_msg void OnButtonFtpPut();
afx_msg void OnLocRefresh();
afx_msg void OnFtpRefresh();
afx_msg void OnGoFtp();
afx_msg void OnLocalhostShowDICOMonly();
afx_msg void OnUpdateLocalhostShowDICOMonly(CCmdUI* pCmdUI);
afx_msg void OnRemotehostFilterDICOMfiles();
afx_msg void OnUpdateRemotehostFilterDICOMfiles(CCmdUI* pCmdUI);
afx_msg void OnRemotehostShowDICOMonly();
afx_msg void OnUpdateRemotehostShowDICOMonly(CCmdUI* pCmdUI);
afx_msg void OnLocalhostDelete();
afx_msg void OnRemotehostDelete();
afx_msg void OnLocalhostRename();
afx_msg void OnRemotehostRename();
afx_msg void OnRclickList(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnLocalhostCreateNewDirectory();
//}}AFX_MSG

```

```

afx_msg void OnRemoteHostCreateNewDirectory();
afx_msg void OnRemoteHostOpen();
afx_msg void OnLocalhostOpen();
afx_msg void OnUpdateLocalhostOpen(CCmdUI* pCmdUI);
afx_msg void OnUpdateRemoteHostOpen(CCmdUI* pCmdUI);
//}}AFX_MSG
afx_msg LRESULT OnKickIdle(WPARAM, LPARAM);
DECLARE_MESSAGE_MAP()

private:
void OpenFile_or_Directory(bool local, int item);
bool CreateNewDirectory(CListCtrl &myList, bool is_local);
bool RenameFile_or_Directory(CListCtrl & myList, bool is_local);
bool DeleteFile_or_Directory(CListCtrl & myList, bool is_local);
bool m_DCMonly_FTP;
bool FileTransfer(CString ftp_name, CString loc_name, bool to_local, int size=-1);
int FindName(CString fname, CListCtrl & myList);
void DICOM_Filter(CListCtrl & myList, bool ftp);
bool resetFTP();
CString m_ParentDirectory_FTP;
void deleteFTP();
bool resetFTP(CString host, CString logon, CString pwd);
bool m_FilterFTP;
bool m_DCMonly_loc;
CString m_INpassword;
CString m_INlogon;
CFtpConnection* m_pFTPConnection;
CInternetSession m_InSession;
void OpenedFilesListRemove(CString fullname);
bool OpenedFilesListFind(CString fullname);
void OpenedFilesListAdd(CString fullname);
CMap<CString,LPCSTR,int, int&> m_Opened;
CString m_TempFile;
CString m_ParentDirectory;
void SetItemImageState(int nitem, int state, CListCtrl & myList);
int GetItemImageState(int nitem, CListCtrl & myList);
CImageList m_ImageList;
void SortByColumn(int col, CListCtrl & myList);
int FindFiles(CListCtrl & myList, bool ftp, CString directory="");
int GetSelectedPosition(bool local);
};

//{ {AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FILEBROWSER_H__B5F74D80_CC60_11D2_9614_00105A21774F__INCLUDED_)

```

```

        filter.Format("%s %s", and, fName, fValue);
    }
    m_strFilter += filter;
}

void ODBCTableSet::AndFilter(CString fName, DateTimeSegment &dValue,
                             const BYTE dFormat)
{
    CString filter("");
    CString and = (m_strFilter == "" ? "" : " AND");
    if(dFormat==DateTime::DateFormat)    // Date
    {
        int nStart = dValue.GetStart().GetNumericDate();
        int nEnd   = dValue.GetEnd().GetNumericDate();

        if(nStart==nEnd)    // exact date
        {
            if(nStart<0)    return;
            filter.Format("%s %s=%d ", and, fName, nStart);
            m_strFilter += filter;
            return;
        }

        // We have interval
        if(nStart<0)    nStart=0;
        if(nEnd<0)
        {
            filter.Format("%s %s>=%d ", and, fName, nStart);
        }
        else
        {
            filter.Format("%s %s BETWEEN %d AND %d ", and, fName,
                           nStart, nEnd);
        }
        m_strFilter += filter;
        return;
    }
    else if(dFormat==DateTime::TimeFormat)    // Time
    {
        double nStart = dValue.GetStart().GetNumericTime();
        double nEnd   = dValue.GetEnd().GetNumericTime();

        if(nStart==nEnd)    // exact date
        {
            if(nStart<0)    return;
            filter.Format("%s %s=%f ", and, fName, nStart);
            m_strFilter += filter;
            return;
        }

        // We have interval
        if(nStart<0)    nStart=0;
        if(nEnd<0)
        {
            filter.Format("%s %s>=%f ", and, fName, nStart);
        }
        else
        {
            filter.Format("%s %s BETWEEN %f AND %f ", and, fName,
                           nStart, nEnd);
        }
        m_strFilter += filter;
        return;
    }
    return;
}

/*****
 *
 *   Add a record to the database, if it did not exist,
 *   or update record empty fields, if the record already existed
 *
 *****/
bool ODBCTableSet::AddOrUpdate(DICOMRecord &dr)
{

```

```

TRY
{
    if(IsOpen())        Close();
    if(!Open(CRecordset::dynaset, (LPCSTR)GetDefaultQuery(dr))) return false;
    if(!CanUpdate())    {    Close(); return false; }
    if(IsEOF()) // need to insert new record
    {
        AddNew();
        if(!SetFromDICOMRecord(dr)) {    Close(); return false; }
        if(!Update())    {    Close(); return false; }
    }
    else                // need to update empty fields in existing record
    {
        Edit();
        UpdateFromDICOMRecord(dr);
        Update();
    }
    Close();
    return true;
}
CATCH(CException, e)
{
    #ifdef _DEBUG
    e->ReportError();
    #endif
    return false;
}
END_CATCH;
return false;
}
/*****
*
* Write data into DICOMObject via DICOMRecord
*
*****/
void ODBCTableSet::WriteIntoDICOMObject(DICOMObject &dob, DICOMObject *dob_mask)
{
    DICOMRecord dr;
    WriteIntoDICOMRecord(dr);
    dr.WriteIntoDICOMObject(dob, dob_mask);
}
//
//
// ODBCPatientSet
//
IMPLEMENT_DYNAMIC(ODBCPatientSet, CRecordset)

ODBCPatientSet::ODBCPatientSet(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{AFX_FIELD_INIT(ODBCPatientSet)
    ClearSet();
    m_nFields = 4;
    //}}AFX_FIELD_INIT
}

/*****
*
* Defining SQL parameters
*
*****/
CString ODBCPatientSet::GetDefaultQuery(DICOMRecord& dr)
{
    CString query;
    query.Format("SELECT * FROM [Patient] WHERE PatientID='%s'",
        ::Trim(dr.GetPatientID()));
    return query;
}

CString ODBCPatientSet::GetDefaultSQL()
{
    return _T("[Patient]");
}

```



```

}
void ODBCPatientSet::SetFilter(DICOMRecord &dr)
{
    m_strFilter = "";

    // Add whatever is known from DICOMRecord. If the primary key
    // is known for a table, do not use other table fields
    AndFilter("PatientID", dr.GetPatientID());
    if(!::IsUniqueString(dr.GetPatientID()))
    {
        AndFilter("PatientName", dr.GetPatientName());
        AndFilter("PBirthDate", dr.GetPBirthDate(), DateTime::DateFormat);
        AndFilter("PBirthTime", dr.GetPBirthTime(), DateTime::TimeFormat);
    }
}

```

```

void ODBCPatientSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(ODBCPatientSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[PatientID]"), m_PatientID);
    RFX_Text(pFX, _T("[PatientName]"), m_PatientName);
    RFX_Double(pFX, _T("[PBirthTime]"), m_PBirthTime);
    RFX_Long(pFX, _T("[PBirthDate]"), m_PBirthDate);
    //}}AFX_FIELD_MAP
}

```

```

/*****
 *
 * Clear patient recordset data
 *
 *****/
void ODBCPatientSet::ClearSet()
{
    m_PatientID = _T("");    m_PatientName = _T("");
    m_PBirthTime = -1.0;    m_PBirthDate = -1;
}

```

```

/*****
 *
 * Exchanging data with DICOMRecord
 *
 *****/
void ODBCPatientSet::UpdateFromDICOMRecord(DICOMRecord &dr)

```

```

{
    char* s;
    if(m_PatientName=="")
    {
        s=dr.GetPatientName();
        if(!::IsEmptyString(s)) m_PatientName = ::Trim(s);
    }
    if(m_PBirthDate <= 0)
    {
        int nDate = dr.GetPBirthDate().GetStart().GetNumericDate();
        if(nDate > 0) m_PBirthDate = nDate;
    }
    if(m_PBirthTime < 0)
    {
        double dTime = dr.GetPBirthDate().GetStart().GetNumericTime();
        if(dTime >= 0) m_PBirthTime=dTime;
    }
}

```

```

bool ODBCPatientSet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetPatientID()))    return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_PatientID = ::Trim(dr.GetPatientID());
    return true;
}

```

```

void ODBCPatientSet::WriteIntoDICOMRecord(DICOMRecord &dr)

```

```

{
    dr.SetRecord((char*)(LPCTSTR)m_PatientID, (char*)(LPCSTR)m_PatientName,
        m_PBirthDate, m_PBirthTime, NULL,
        NULL, NULL,
        NULL, -1, -1.0,
        NULL, NULL,
        NULL, NULL,
        NULL, NULL);
}

////////////////////////////////////
//
// ODBCStudySet
//
////////////////////////////////////
IMPLEMENT_DYNAMIC(ODBCStudySet, CRecordset)

ODBCStudySet::ODBCStudySet(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{AFX_FIELD_INIT(ODBCStudySet)
    ClearSet();
    m_nFields = 7;
    //}AFX_FIELD_INIT
}

void ODBCStudySet::ClearSet()
{
    m_PatientID = _T("");          m_StudyInstUID = _T("");
    m_StudyID = _T("");            m_AccessionNumber = _T("");
    m_StudyTime = -1.0;            m_StudyDate = -1;
    m_StudyImagesNum = _T("");
}

*****
Defining SQL parameters
*****/
CString ODBCStudySet::GetDefaultQuery(DICOMRecord &dr)
{
    CString query;
    query.Format("SELECT * FROM [Study] WHERE StudyInstUID='%s'",
        ::Trim(dr.GetStudyInstUID()));
    return query;
}

CString ODBCStudySet::GetDefaultSQL()
{
    return _T("[Study]");
}

void ODBCStudySet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";

    // Add whatever is known from DICOMRecord. If the primary key
    // is known for a table, do not use other table fields
    AndFilter("StudyInstUID", dr.GetStudyInstUID());
    if(!::IsUniqueString(dr.GetStudyInstUID()))
    {
        AndFilter("PatientID", dr.GetPatientID());
        AndFilter("StudyID", dr.GetStudyID());
        AndFilter("StudyImagesNum", dr.GetStudyImagesNum());
        AndFilter("AccessionNumber", dr.GetAccessionNumber());
        AndFilter("StudyDate", dr.GetStudyDate(), DateTime::DateFormat);
        AndFilter("StudyTime", dr.GetStudyTime(), DateTime::TimeFormat);
    }
}

void ODBCStudySet::DoFieldExchange(CFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(ODBCStudySet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[PatientID]"), m_PatientID);
    RFX_Text(pFX, _T("[StudyInstUID]"), m_StudyInstUID);
    RFX_Text(pFX, _T("[StudyID]"), m_StudyID);
    RFX_Text(pFX, _T("[AccessionNumber]"), m_AccessionNumber);
    RFX_Double(pFX, _T("[StudyTime]"), m_StudyTime);
}

```

```

RFX_Long(pFX, _T("[StudyID]"), m_StudyDate);
RFX_Text(pFX, _T("[StudyImagesNum]"), m_StudyImagesNum);
//}}AFX_FIELD_MAP
}
/*****
*
*   Exchanging data with DICOMRecord
*
*****/
void ODBCStudySet::UpdateFromDICOMRecord(DICOMRecord &dr)
{
    char* s;
    if(m_PatientID=="")
    {
        s=dr.GetPatientID();
        if(!::IsEmptyString(s)) m_PatientID = ::Trim(s);
    }
    if(m_StudyID=="")
    {
        s=dr.GetStudyID();
        if(!::IsEmptyString(s)) m_StudyID = ::Trim(s);
    }
    if(m_AccessionNumber=="")
    {
        s = dr.GetAccessionNumber();
        if(!::IsEmptyString(s)) m_AccessionNumber = ::Trim(s);
    }
    if(m_StudyImagesNum=="")
    {
        s = dr.GetStudyImagesNum();
        if(!::IsEmptyString(s)) m_StudyImagesNum = ::Trim(s);
    }
    if(m_StudyDate <= 0)
    {
        int nDate = dr.GetStudyDate().GetStart().GetNumericDate();
        if(nDate > 0) m_StudyDate = nDate;
    }
    if(m_StudyTime < 0)
    {
        double dTime = dr.GetStudyTime().GetStart().GetNumericTime();
        if(dTime >= 0) m_StudyTime=dTime;
    }
}

bool ODBCStudySet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetStudyInstUID())) return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_StudyInstUID = ::Trim(dr.GetStudyInstUID());
    return true;
}

void ODBCStudySet::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord((char*)(LPCSTR)m_PatientID,NULL,
        -1, -1.0,(char*)(LPCSTR)m_StudyInstUID,
        (char*)(LPCSTR)m_StudyID,(char*)(LPCSTR)m_AccessionNumber,
        (char*)(LPCSTR)m_StudyImagesNum,m_StudyDate,m_StudyTime,
        NULL,NULL,
        NULL,NULL,
        NULL,NULL);
}

//
//   ODBCStudySet
//
IMPLEMENT_DYNAMIC(ODBCStudySet, CRecordset)

ODBCStudySet::ODBCStudySet(CDatabase* pDb)
: ODBCTableSet(theApp._DataBase.GetCDatabasePtr())
{
    //{{AFX_FIELD_INIT(ODBCStudySet)

```

```

    ClearSet();
    m_nFields = 4;
    //}}AFX_FIELD_INIT
}
void ODBCSeriesSet::ClearSet()
{
    m_StudyInstUID = _T("");    m_SeriesInstUID = _T("");
    m_Modality = _T("");    m_SeriesNum = _T("");
}
/*****
*
*   Defining SQL parameters
*
*****/
CString ODBCSeriesSet::GetDefaultQuery(DICOMRecord &dr)
{
    CString query;
    query.Format("SELECT * FROM [Series] WHERE SeriesInstUID='%s'",
        ::Trim(dr.GetSeriesInstUID()));
    return query;
}
CString ODBCSeriesSet::GetDefaultSQL()
{
    return _T("[Series]");
}

void ODBCSeriesSet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";

    // Add whatever is known from DICOMRecord. If the primary key
    // is known for a table, do not use other table fields
    AndFilter("SeriesInstUID", dr.GetSeriesInstUID());
    if(!::IsUniqueString(dr.GetSeriesInstUID()))
    {
        AndFilter("StudyInstUID", dr.GetStudyInstUID());
        AndFilter("Modality", dr.GetModality());
        AndFilter("SeriesNum", dr.GetSeriesNum());
    }
}

void ODBCSeriesSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(ODBCSeriesSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[StudyInstUID]"), m_StudyInstUID);
    RFX_Text(pFX, _T("[SeriesInstUID]"), m_SeriesInstUID);
    RFX_Text(pFX, _T("[Modality]"), m_Modality);
    RFX_Text(pFX, _T("[SeriesNum]"), m_SeriesNum);
    //}}AFX_FIELD_MAP
}
/*****
*
*   Exchanging data with DICOMRecord
*
*****/
void ODBCSeriesSet::UpdateFromDICOMRecord(DICOMRecord &dr)
{
    char* s;
    //static UINT alt=0;
    if(m_StudyInstUID=="")
    {
        s = dr.GetStudyInstUID();
        if(!::IsEmptyString(s)) m_StudyInstUID = ::Trim(s);
    }
    /*
    else
    {
        // Consistency check
        CString prkey = ::Trim(dr.GetSeriesInstUID());
        if( m_SeriesInstUID == prkey &&
            m_StudyInstUID != dr.GetStudyInstUID())
        {

```

```

        alt++;
        m_SeriesInstUID.Format("%s.%d", prkey, alt);
        sprintf(dr.GetSeriesInstUID(), "%s", m_SeriesInstUID.Left(64));
    }
}
*/
if(m_Modality=="")
{
    s = dr.GetModality();
    if(!::IsEmptyString(s)) m_Modality = ::Trim(s);
}
if(m_SeriesNum=="")
{
    s = dr.GetSeriesNum();
    if(!::IsEmptyString(s)) m_SeriesNum = ::Trim(s);
}
}
bool ODBCSeriesSet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetSeriesInstUID()))    return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_SeriesInstUID = ::Trim(dr.GetSeriesInstUID());
    return true;
}
void ODBCSeriesSet::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord(NULL, NULL,
        -1, -1.0, (char*) (LPCSTR)m_StudyInstUID,
        NULL, NULL,
        NULL, -1, -1.0,
        (char*) (LPCSTR)m_SeriesInstUID, (char*) (LPCSTR)m_Modality,
        (char*) (LPCSTR)m_SeriesNum, NULL,
        NULL, NULL);
}
///////////////////////////////////////////////////
// ODBCImageSet
///////////////////////////////////////////////////
IMPLEMENT_DYNAMIC(ODBCImageSet, CRecordset)
ODBCImageSet::ODBCImageSet(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{
    //{AFX_FIELD_INIT(ODBCImageSet)
    ClearSet();
    m_nFields = 4;
    //}AFX_FIELD_INIT
}
void ODBCImageSet::ClearSet()
{
    m_SeriesInstUID = _T("");    m_SOPInstUID = _T("");
    m_ImageNum = _T("");    m_Filename = _T("");
}
/*****
*
*   Defining SQL parameters
*
*****/
CString ODBCImageSet::GetDefaultQuery(DICOMRecord &dr)
{
    CString query;
    query.Format("SELECT * FROM [Image] WHERE SOPInstUID='%s'",
        ::Trim(dr.GetSOPInstUID()));
    return query;
}
CString ODBCImageSet::GetDefaultSQL()
{
    return _T("[Image]");
}
void ODBCImageSet::SetFindFilter(DICOMRecord &dr)
{
    m_strFilter = "";
}

```

```

// Add whatever is known from DICOMRecord. If the primary key
// is known for a table, do not use other table fields
AndFilter("SOPInstUID", dr.GetSOPInstUID());
if(!::IsUniqueString(dr.GetSOPInstUID()))
{
    AndFilter("SeriesInstUID", dr.GetSeriesInstUID());
    AndFilter("ImageNum", dr.GetImageNum());
    AndFilter("Filename", dr.GetFileName());
}
}

void ODBCImageSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{AFX_FIELD_MAP(ODBCImageSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[SeriesInstUID]"), m_SeriesInstUID);
    RFX_Text(pFX, _T("[SOPInstUID]"), m_SOPInstUID);
    RFX_Text(pFX, _T("[ImageNum]"), m_ImageNum);
    RFX_Text(pFX, _T("[Filename]"), m_Filename);
    //}AFX_FIELD_MAP
}

/*****
 *
 *   Exchanging data with DICOMRecord
 *
 *****/
void ODBCImageSet::UpdateFromDICOMRecord(DICOMRecord &dr)
{
    char* s;
    if(m_SeriesInstUID=="")
    {
        s = dr.GetSeriesInstUID();
        if(!::IsEmptyString(s)) m_SeriesInstUID = ::Trim(s);
    }
    if(m_ImageNum=="")
    {
        s = dr.GetImageNum();
        if(!::IsEmptyString(s)) m_ImageNum = ::Trim(s);
    }
    if(m_Filename=="")
    {
        s = dr.GetFileName();
        if(!::IsEmptyString(s)) m_Filename = ::Trim(s);
    }
}

bool ODBCImageSet::SetFromDICOMRecord(DICOMRecord &dr)
{
    if(!::IsUniqueString(dr.GetSOPInstUID())) return false;
    ClearSet();
    UpdateFromDICOMRecord(dr);
    m_SOPInstUID = ::Trim(dr.GetSOPInstUID());
    return true;
}

void ODBCImageSet::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord(NULL, NULL,
        -1, -1.0, NULL,
        NULL, NULL,
        NULL, -1, -1.0,
        (char*)(LPCSTR)m_SeriesInstUID, NULL,
        NULL, (char*)(LPCSTR)m_SOPInstUID,
        (char*)(LPCSTR)m_ImageNum, (char*)(LPCSTR)m_Filename);
}

////////////////////////////////////
// ODBC4Set

IMPLEMENT_DYNAMIC(ODBC4Set, CRecordset)

ODBC4Set::ODBC4Set(CDatabase* pdb)
: ODBCTableSet(theApp.app_DataBase.GetCDatabasePtr())
{

```

```

//{{AFX_FIELD_INIT(ODBC4Set)
ClearSet();
m_nFields = 19;
//}}AFX_FIELD_INIT
}

void ODBC4Set::ClearSet()
{
    m_SeriesInstUID = _T("");    m_SOPInstUID = _T("");
    m_ImageNum = _T("");        m_Filename = _T("");
    m_PatientID = _T("");        m_PatientName = _T("");
    m_PBirthTime = -1.0;        m_PBirthDate = -1;
    m_StudyInstUID = _T("");    m_SeriesInstUID2 = _T("");
    m_Modality = _T("");        m_SeriesNum = _T("");
    m_PatientID2 = _T("");        m_StudyInstUID2 = _T("");
    m_StudyID = _T("");        m_AccessionNumber = _T("");
    m_StudyTime = -1.0;        m_StudyDate = -1;
    m_StudyImagesNum = _T("");
}

CString ODBC4Set::GetDefaultSQL()
{
    return _T("[Image].[Patient].[Series].[Study]");
}

/*****
*
*   Redefines pure virtual from the base class. Unused
*
*****/
CString ODBC4Set::GetDefaultQuery(DICOMRecord &dr)
{
    m_strFilter = "[Patient].[PatientID]=[Study].[PatientID] AND "
        "[Study].[StudyInstUID]=[Series].[StudyInstUID] AND "
        "[Series].[SeriesInstUID]=[Image].[SeriesInstUID] ";
    return m_strFilter;
}

void ODBC4Set::DoFieldExchange(CFieldExchange* pFX)
{
    //{{{AFX_FIELD_MAP(ODBC4Set)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[Image].[SeriesInstUID]"), m_SeriesInstUID);
    RFX_Text(pFX, _T("[SOPInstUID]"), m_SOPInstUID);
    RFX_Text(pFX, _T("[ImageNum]"), m_ImageNum);
    RFX_Text(pFX, _T("[Filename]"), m_Filename);
    RFX_Text(pFX, _T("[Patient].[PatientID]"), m_PatientID);
    RFX_Text(pFX, _T("[PatientName]"), m_PatientName);
    RFX_Double(pFX, _T("[PBirthTime]"), m_PBirthTime);
    RFX_Long(pFX, _T("[PBirthDate]"), m_PBirthDate);
    RFX_Text(pFX, _T("[Series].[StudyInstUID]"), m_StudyInstUID);
    RFX_Text(pFX, _T("[Series].[SeriesInstUID]"), m_SeriesInstUID2);
    RFX_Text(pFX, _T("[Modality]"), m_Modality);
    RFX_Text(pFX, _T("[SeriesNum]"), m_SeriesNum);
    RFX_Text(pFX, _T("[Study].[PatientID]"), m_PatientID2);
    RFX_Text(pFX, _T("[Study].[StudyInstUID]"), m_StudyInstUID2);
    RFX_Text(pFX, _T("[StudyID]"), m_StudyID);
    RFX_Text(pFX, _T("[AccessionNumber]"), m_AccessionNumber);
    RFX_Double(pFX, _T("[StudyTime]"), m_StudyTime);
    RFX_Long(pFX, _T("[StudyDate]"), m_StudyDate);
    RFX_Text(pFX, _T("[StudyImagesNum]"), m_StudyImagesNum);
    //}}}AFX_FIELD_MAP
}

/*****
*
*   Set complete WHERE find filter
*
*****/
void ODBC4Set::SetFindFilter(DICOMRecord &dr)
{
    // Relational constraint
    m_strFilter = "[Patient].[PatientID]=[Study].[PatientID] AND "
        "[Study].[StudyInstUID]=[Series].[StudyInstUID] AND "

```

```

        "[Series].[SeriesInstUID]=[Image].[SeriesInstUID]";

// Add whatever is known from DICOMRecord. If the primary key
// is known for a table, do not use other table fields
AndFilter("[Patient].[PatientID]", dr.GetPatientID());
if(!::IsUniqueString(dr.GetPatientID()))
{
    AndFilter("PatientName",dr.GetPatientName());
    AndFilter("PBirthDate",dr.GetPBirthDate(), DateTime::DateFormat);
    AndFilter("PBirthTime",dr.GetPBirthTime(), DateTime::TimeFormat);
}

AndFilter("[Study].[StudyInstUID]",dr.GetStudyInstUID());
if(!::IsUniqueString(dr.GetStudyInstUID()))
{
    AndFilter("StudyID",dr.GetStudyID());
    AndFilter("StudyImagesNum",dr.GetStudyImagesNum());
    AndFilter("AccessionNumber",dr.GetAccessionNumber());
    AndFilter("StudyDate", dr.GetStudyDate(), DateTime::DateFormat);
    AndFilter("StudyTime", dr.GetStudyTime(), DateTime::TimeFormat);
}

AndFilter("[Series].[SeriesInstUID]",dr.GetSeriesInstUID());
if(!::IsUniqueString(dr.GetSeriesInstUID()))
{
    AndFilter("Modality", dr.GetModality());
    AndFilter("SeriesNum", dr.GetSeriesNum());
}

AndFilter("SOPInstUID",dr.GetSOPInstUID());
if(!::IsUniqueString(dr.GetSOPInstUID()))
{
    AndFilter("ImageNum", dr.GetImageNum());
    AndFilter("Filename", dr.GetFileName());
}

*****
Put data into DICOM record
*****
void ODBC4Set::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    dr.SetRecord((char*)(LPCSTR)m_PatientID,(char*)(LPCSTR)m_PatientName,
        m_PBirthDate, m_PBirthTime,(char*)(LPCSTR)m_StudyInstUID,
        (char*)(LPCSTR)m_StudyID,(char*)(LPCSTR)m_AccessionNumber,
        (char*)(LPCSTR)m_StudyImagesNum,m_StudyDate,m_StudyTime,
        (char*)(LPCSTR)m_SeriesInstUID,(char*)(LPCSTR)m_Modality,
        (char*)(LPCSTR)m_SeriesNum,(char*)(LPCSTR)m_SOPInstUID,
        (char*)(LPCSTR)m_ImageNum,(char*)(LPCSTR)m_Filename);
}

/*****
*
* Make sure that several different records on one level
* cannot share records on lower level
*
*****/
bool ODBC4Set::HasAliasRecords(DICOMRecord &dr)
{
    CString p=::Trim(dr.GetPatientID());
    CString st=::Trim(dr.GetStudyInstUID());
    CString sr=::Trim(dr.GetSeriesInstUID());
    CString im=::Trim(dr.GetSOPInstUID());
    CString cons;
    cons.Format(
        " AND ( ( [Patient].[PatientID]<>'%s' AND [Study].[StudyInstUID]='%s' ) OR "
        "( [Study].[StudyInstUID]<>'%s' AND [Series].[SeriesInstUID]='%s' ) OR "
        "( [Series].[SeriesInstUID]<>'%s' AND [Image].[SOPInstUID]='%s' ) )", p, st,
        st, sr, sr, im);
    m_strFilter = "[Patient].[PatientID]=[Study].[PatientID] AND "
        "[Study].[StudyInstUID]=[Series].[StudyInstUID] AND "
        "[Series].[SeriesInstUID]=[Image].[SeriesInstUID] ";
    m_strFilter += cons;
}

```



```
bool aliased = true;
TRY
{
    Open();
    aliased = (IsEOF()==FALSE);
    Close();
}
CATCH(CException, e)
{
#ifdef DEBUG
    e->ReportError();
#endif
}
END_CATCH;
return aliased;
```

```

#ifndef AFX_QUERYRETRIEVE_H_INCLUDED_
#define AFX_QUERYRETRIEVE_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// QueryRetrieve.h : header file
//

#include "dqrcontrol.h" // Added by ClassView
#include "..\LogFile.h" // Added by ClassView

////////////////////////////////////

// QueryRetrieve dialog

class QueryRetrieve : public CDialog
{
// Construction
public:
    void SwitchAppearance();
    void DoModeless(CWnd* pParent=NULL);
    void OnBeginDrag(CListCtrl* pList, NMHDR* pNMHDR);
    bool InitializeQueryRetrieve(LogFile* ptrClientLog,
                                LogFile* ptrServerLog, DICOMDatabase* ptrDB);
    QueryRetrieve(CWnd* pParent = NULL); // constructor
    ~QueryRetrieve(); // destructor

// Dialog Data
    //{AFX_DATA(QueryRetrieve)
    enum { IDD = IDD_DIALOG_QUERY_RETRIEVE };
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(QueryRetrieve)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{AFX_MSG(QueryRetrieve)
    afx_msg void OnQueryRetrieve_AESetup();
    virtual BOOL OnInitDialog();
    afx_msg void OnParametersPriorityHigh();
    afx_msg void OnParametersPriorityLow();
    afx_msg void OnParametersPriorityNormal();
    afx_msg void OnUpdateParametersPriority(CCmdUI* pCmdUI);
    afx_msg void OnQueryRetrieveClientLog();
    afx_msg void OnQueryRetrieveServerLog();
    afx_msg void OnClose();
    virtual void OnOK();
    afx_msg void OnUpdateQueryRetrieveClientLog(CCmdUI* pCmdUI);
    afx_msg void OnUpdateQueryRetrieveServerLog(CCmdUI* pCmdUI);
    afx_msg void OnQueryRetrieveClearAllLogs();
    afx_msg void OnMenuSelect(UINT nItemID, UINT nFlags, HMENU hSysMenu);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnHide();
    afx_msg void OnExit();
    afx_msg void OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized);
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnServicesShowRemoteTasks();
    afx_msg void OnUpdateServicesShowRemoteTasks(CCmdUI* pCmdUI);
    afx_msg void OnServicesTaskScheduling();
    afx_msg void OnUpdateServicesTaskScheduling(CCmdUI* pCmdUI);
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    bool qr_UpdateMenu;
    HWND qr_HWND;
    LogFile *qr_ptrClientLog, *qr_ptrServerLog;
    ApplicationEntityList* qr_AEarray;

```

```

DQRControl      DQRCtrlRemote, qr_DQRCtrlLocal;
CWnd*           bDragWnd;
CImageList*     qr_pDragImage;

void            OnCancel();
void            UpdateMenu (CMenu* pMenu);
BOOL            UpdateData( BOOL bSaveAndValidate=TRUE);
CImageList*     CreateDragImageEx(CListCtrl *pList, LPPOINT lpPoint);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_QUERYRETRIEVE_H_INCLUDED_)

```

```
// QueryRetrieve.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\Resource.h"
#include "QueryRetrieve.h"
#include "AEOptions_Dialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// QueryRetrieve dialog
```

```
QueryRetrieve::QueryRetrieve(CWnd* pParent /*=NULL*/)
: CDialog(QueryRetrieve::IDD, pParent)
```

```
{
    //{{AFX_DATA_INIT(QueryRetrieve)
    //}}AFX_DATA_INIT
    qr_UpdateMenu=false;
    qr_HWND = NULL;
    qr_AEarray = NULL;
    qr_pDragImage = NULL;
    qr_pDragWnd = NULL;
}
```

```
QueryRetrieve::~QueryRetrieve()
```

```
void QueryRetrieve::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(QueryRetrieve)
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(QueryRetrieve, CDialog)
```

```
    //{{AFX_MSG_MAP(QueryRetrieve)
    ON_COMMAND(ID_QUERYRETRIEVE_AESETUP, OnQueryRetrieve_AESetup)
    ON_COMMAND(ID_PARAMETERS_PRIORITY_HIGH, OnParametersPriorityHigh)
    ON_COMMAND(ID_PARAMETERS_PRIORITY_LOW, OnParametersPriorityLow)
    ON_COMMAND(ID_PARAMETERS_PRIORITY_NORMAL, OnParametersPriorityNormal)
    ON_UPDATE_COMMAND_UI(ID_PARAMETERS_PRIORITY_HIGH, OnUpdateParametersPriority)
    ON_COMMAND(ID_QUERYRETRIEVE_CLIENTLOG, OnQueryRetrieveClientLog)
    ON_COMMAND(ID_QUERYRETRIEVE_SERVERLOG, OnQueryRetrieveServerLog)
    ON_WM_CLOSE()
    ON_UPDATE_COMMAND_UI(ID_QUERYRETRIEVE_CLIENTLOG, OnUpdateQueryRetrieveClientLog)
    ON_UPDATE_COMMAND_UI(ID_QUERYRETRIEVE_SERVERLOG, OnUpdateQueryRetrieveServerLog)
    ON_COMMAND(ID_QUERYRETRIEVE_CLEARALLLOGS, OnQueryRetrieveClearAllLogs)
    ON_WM_MENUSELECT()
    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
    ON_WM_ACTIVATE()
    ON_WM_SYSCOMMAND()
    ON_COMMAND(ID_SERVICES_SHOWREMOTE TASKS, OnServicesShowRemoteTasks)
    ON_UPDATE_COMMAND_UI(ID_SERVICES_SHOWREMOTE TASKS, OnUpdateServicesShowRemoteTasks)
    ON_COMMAND(ID_SERVICES_TASKSCHEDULING, OnServicesTaskScheduling)
    ON_UPDATE_COMMAND_UI(ID_PARAMETERS_PRIORITY_LOW, OnUpdateParametersPriority)
    ON_UPDATE_COMMAND_UI(ID_PARAMETERS_PRIORITY_NORMAL, OnUpdateParametersPriority)
    ON_UPDATE_COMMAND_UI(ID_SERVICES_TASKSCHEDULING, OnUpdateServicesTaskScheduling)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// QueryRetrieve message handlers
```

```
/*
*****
*/
```

```

*
*   Alter Query/Retrieve mode dialog
*
*****/
void QueryRetrieve::DoModeless(CWnd* pParent)
{
    if(GetSafeHwnd())
    {
        ShowWindow(SW_SHOWNORMAL);
        return;
    }
    Create(IDD_DIALOG_QUERY_RETRIEVE, pParent);
    ShowWindow(SW_SHOWNORMAL);
    qr_UpdateMenu=true;
}

void QueryRetrieve::OnClose() { OnHide(); }
void QueryRetrieve::OnOK() { OnHide(); }
void QueryRetrieve::OnCancel() { OnHide(); }
void QueryRetrieve::OnHide()
{
    if(!qr_HWND) return;
    ShowWindow(SW_MINIMIZE);
}

void QueryRetrieve::OnExit()
{
    if(!qr_HWND) return;
    CString msg("Do you want to terminate DICOM Query/Retrieve ?\n");
    msg += CString("If Yes, you will have to restart DCM\n");
    msg += CString("to enable Query/Retrieve again.");
    if( AfxMessageBox(msg, MB_ICONQUESTION | MB_YESNO) == IDNO ) return;
    DestroyWindow();
}

void QueryRetrieve::SwitchAppearance()
{
    if(GetSafeHwnd() == NULL) // Display
    {
        DoModeless(AfxGetMainWnd());
    }
    else // Maximize / Miinimize
    {
        if(IsIconic()) ShowWindow(SW_RESTORE);
        else ShowWindow(SW_MINIMIZE);
        UpdateWindow(); // Make sure it redraws the window
    }
}

void QueryRetrieve::OnSysCommand(UINT nID, LPARAM lParam)
{
    if(nID == SC_MAXIMIZE || nID == SC_ZOOM) ShowWindow(SW_RESTORE);
    else CDialog::OnSysCommand(nID, lParam);
    UpdateWindow(); // Make sure it redraws the window
}

void QueryRetrieve::OnActivate(UINT nState, CWnd* pWndOther, BOOL bMinimized)
{
    CDialog::OnActivate(nState, pWndOther, bMinimized);
    if(nState == WA_INACTIVE && pWndOther==AfxGetMainWnd())
        ShowWindow(SW_MINIMIZE);
    UpdateWindow(); // Make sure it redraws the window
}

/*****
*
*   Run AE setup
*
*****/
void QueryRetrieve::OnQueryRetrieve_AESetup()
{
    AEOptions_Dialog aeo_dialog;
    int n = aeo_dialog.DoModal(qr_AEarray);
    if(n>=0)
    {

```

```
qr_DQRCtrlRemote.LoadShiveList(n);
qr_DQRCtrlLocal.LoadShiveList(n);
}
```

```
/******
```

```
* Handle menu updates
```

```
*****/
```

```
void QueryRetrieve::OnMenuSelect(UINT nItemID, UINT nFlags, HMENU hSysMenu)
```

```
{
    CDialog::OnMenuSelect(nItemID, nFlags, hSysMenu);
    if(qr_UpdateMenu) UpdateMenu(GetMenu());
    qr_UpdateMenu = false;
}
```

```
void QueryRetrieve::UpdateMenu(CMenu *pMenu)
```

```
{
    CCmdUI cmdUI;
    if (!pMenu) return;
    for (UINT n = 0; n < pMenu->GetMenuItemCount(); ++n)
    {
        CMenu* pSubMenu = pMenu->GetSubMenu(n);
        if (pSubMenu) UpdateMenu(pSubMenu); // recursive call
        else
        {
            cmdUI.m_nIndexMax = pMenu->GetMenuItemCount();
            for (UINT i = 0; i < cmdUI.m_nIndexMax; ++i)
            {
                cmdUI.m_nIndex = i;
                cmdUI.m_nID = pMenu->GetMenuItemID(i);
                cmdUI.m_pMenu = pMenu;
                cmdUI.DoUpdate(this, FALSE); // call handler
            }
        }
    }
}
```

```
*****
```

```
Change priority
```

```
*****/
```

```
void QueryRetrieve::OnParametersPriorityHigh()
```

```
{
    qr_DQRCtrlRemote.SetPriority(ServiceClass::HighPriority);
    qr_DQRCtrlLocal.SetPriority(ServiceClass::HighPriority);
}
```

```
void QueryRetrieve::OnParametersPriorityLow()
```

```
{
    qr_DQRCtrlRemote.SetPriority(ServiceClass::LowPriority);
    qr_DQRCtrlLocal.SetPriority(ServiceClass::LowPriority);
}
```

```
void QueryRetrieve::OnParametersPriorityNormal()
```

```
{
    qr_DQRCtrlRemote.SetPriority(ServiceClass::NormalPriority);
    qr_DQRCtrlLocal.SetPriority(ServiceClass::NormalPriority);
}
```

```
void QueryRetrieve::OnUpdateParametersPriority(CCmdUI* pCmdUI)
```

```
{
    BYTE pr;
    switch(pCmdUI->m_nID)
    {
        case ID_PARAMETERS_PRIORITY_HIGH:
            pr=ServiceClass::HighPriority;
            break;
        case ID_PARAMETERS_PRIORITY_NORMAL:
            pr=ServiceClass::NormalPriority;
            break;
        default:
            pr=ServiceClass::LowPriority;
            break;
    }
}
```

```
pCmdUI->SetCheck(qr_DQRCtrlRemote.GetPriority() == pr);
qr_UpdateMenu=true;
}
```

```

/*****
*
*   Start the dialog
*
*****/

```

```
BOOL QueryRetrieve::OnInitDialog()
```

```
{
    CDialog::OnInitDialog();
    qr_HWND=GetSafeHwnd();
    qr_UpdateMenu=true;
    if(!qr_DQRCtrlRemote.DisplayOverControl(IDC_STATIC_REMOTE, this))
        return FALSE;
    if(!qr_DQRCtrlLocal.DisplayOverControl(IDC_STATIC_LOCAL, this))
        return FALSE;
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
```

```

/*****
*
*   Set log files (client and server)
*
*****/

```

```
bool QueryRetrieve::InitializeQueryRetrieve(LogFile* ptrClientLog,
                                           LogFile* ptrServerLog, DICOMDatabase* ptrDB)
```

```
{
    // Set parameters
    if(!ptrClientLog || !ptrServerLog || !ptrDB)
    {
        AfxMessageBox("NULL parameters, cannot initialize Query/Retrieve");
        return false;
    }
    qr_ptrClientLog = ptrClientLog;
    qr_ptrServerLog = ptrServerLog;
    if(! qr_DQRCtrlRemote.CreateDQRControl(qr_ptrClientLog, ptrDB, false) ||
        ! qr_DQRCtrlLocal.CreateDQRControl(qr_ptrClientLog, ptrDB, true))
    {
        return false;
    }
    qr_AEarray = &(ptrDB->db_AElist);
    return true;
}
```

```

/*****
*
*   Show log windows
*
*****/

```

```
void QueryRetrieve::OnQueryRetrieveClientLog()
```

```
{
    if(!qr_ptrClientLog->IsOn())    qr_ptrClientLog->DoModeless("Client Log");
    else                            qr_ptrClientLog->DestroyWindow();
}
```

```
void QueryRetrieve::OnQueryRetrieveServerLog()
```

```
{
    if(!qr_ptrServerLog->IsOn())    qr_ptrServerLog->DoModeless("Server Log");
    else                            qr_ptrServerLog->DestroyWindow();
}
```

```
void QueryRetrieve::OnUpdateQueryRetrieveClientLog(CCmdUI* pCmdUI)
```

```
{
    pCmdUI->SetCheck(qr_ptrClientLog->IsOn());
    qr_UpdateMenu=true;
}
```

```
void QueryRetrieve::OnUpdateQueryRetrieveServerLog(CCmdUI* pCmdUI)
```

```
{
    pCmdUI->SetCheck(qr_ptrServerLog->IsOn());
    qr_UpdateMenu=true;
}
```

```

}
void QueryRetrieve::OnQueryRetrieveClearAllLogs()
{
    qr_ptrClientLog->OnClear();
    qr_ptrServerLog->OnClear();
}

/*****
*
*   Thread-safe UpdateData
*
*****/
BOOL QueryRetrieve::UpdateData(BOOL bSaveAndValidate)
{
    if(qr_HWND) return FromHandle(qr_HWND)->UpdateData(bSaveAndValidate);
    else return FALSE;
}

/*****
*
*   Drag and drop support
*
*****/
CImageList* QueryRetrieve::CreateDragImageEx(CListCtrl *pList, LPPOINT lpPoint)
{
    if (!pList || pList->GetSelectedCount() <= 0) return NULL; //No row selected

    CRect rectSingle, rectComplete(0,0,0,0);

    // Determine List Control Client width size
    pList->GetClientRect(rectSingle);
    int nWidth = rectSingle.Width();

    // Start and Stop index in view area
    int nIndex = pList->GetTopIndex() - 1;
    int nBottomIndex = pList->GetTopIndex() + pList->GetCountPerPage() - 1;
    if (nBottomIndex > (pList->GetItemCount() - 1))
        nBottomIndex = pList->GetItemCount() - 1;

    // Determine the size of the drag image (limited for rows visible and Client width)
    while ((nIndex = pList->GetNextItem(nIndex, LVNI_SELECTED)) != -1)
    {
        if (nIndex > nBottomIndex)
            break;

        pList->GetItemRect(nIndex, rectSingle, LVIR_BOUNDS);

        if (rectSingle.left < 0)
            rectSingle.left = 0;

        if (rectSingle.right > nWidth)
            rectSingle.right = nWidth;

        rectComplete.UnionRect(rectComplete, rectSingle);
    }
    // Minimize drag rectangle width to the size of the first column
    rectComplete.right = rectComplete.left + pList->GetColumnWidth(0);

    CClientDC dcClient(this);
    CDC dcMem;
    CBitmap Bitmap;

    if (!dcMem.CreateCompatibleDC(&dcClient))
        return NULL;

    if (!Bitmap.CreateCompatibleBitmap(&dcClient, rectComplete.Width(), rectComplete.Height()))
        return NULL;

    CBitmap *pOldMemDCBitmap = dcMem.SelectObject(&Bitmap);
    // Use green as mask color
    dcMem.FillSolidRect(0, 0, rectComplete.Width(), rectComplete.Height(), RGB(0,255,0));

```



```

// Paint each DragImage in the DC
nIndex = pList->GetTopIndex() - 1;
while ((nIndex = pList->GetNextItem(nIndex, LVNI_SELECTED)) != -1)
{
    if (nIndex > nBottomIndex)
        break;

    CPoint pt;
    CImageList* pSingleImageList = pList->CreateDragImage(nIndex, &pt);

    if (pSingleImageList)
    {
        pList->GetItemRect(nIndex, rectSingle, LVIR_BOUNDS);
        pSingleImageList->Draw( &dcMem,
                                0,
                                CPoint(rectSingle.left - rectComplete.left,
                                        rectSingle.top - rectComplete.top),
                                ILD_MASK);
        pSingleImageList->DeleteImageList();
        delete pSingleImageList;
    }

    dcMem.SelectObject(pOldMemDCBitmap);
    CImageList* pCompleteImageList = new CImageList;
    pCompleteImageList->Create(rectComplete.Width(), rectComplete.Height(), ILC_COLOR | ILC_MASK,
0, 1);
    pCompleteImageList->Add(&Bitmap, RGB(0, 255, 0)); // Green is used as mask color
    Bitmap.DeleteObject();

    if (lpPoint)
    {
        lpPoint->x = rectComplete.left;
        lpPoint->y = rectComplete.top;
    }

    return pCompleteImageList;
}

void QueryRetrieve::OnBeginDrag(CListCtrl *pList, NMHDR *pNMHDR)
{
    if (!pList || pList->GetSelectedCount() <= 0) return; //No row selected

    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    POINT pt;
    qr_pDragImage = CreateDragImageEx(pList, &pt);

    if (qr_pDragImage == NULL) return;

    qr_pDragWnd = pList;
    CPoint ptStart = pNMListView->ptAction;
    ptStart -= pt;
    qr_pDragImage->BeginDrag(0, ptStart);
    qr_pDragImage->DragEnter(GetDesktopWindow(), pNMListView->ptAction);
    SetCapture();
}

void QueryRetrieve::OnLButtonUp(UINT nFlags, CPoint point)
{
    if (qr_pDragImage && qr_pDragWnd) // In Drag&Drop mode ?
    {
        ::ReleaseCapture();
        qr_pDragImage->DragLeave(GetDesktopWindow());
        qr_pDragImage->EndDrag();

        CPoint pt(point);
        ClientToScreen(&pt);
        CWnd* pDropWnd = WindowFromPoint(pt);
        qr_DQRCtrlLocal.DropOn(qr_pDragWnd, pDropWnd);
        qr_DQRCtrlRemote.DropOn(qr_pDragWnd, pDropWnd);
        qr_pDragImage->DeleteImageList();
        delete qr_pDragImage;
        qr_pDragImage = NULL;
        qr_pDragWnd = NULL;
    }
}

```

```

    CDialog::OnLButtonUp(nFlags, point);
}
void QueryRetrieve::OnMouseMove(UINT nFlags, CPoint point)
{
    if (qr_pDragImage && qr_pDragWnd) // In Drag&Drop mode ?
    {
        CPoint ptDropPoint(point);
        ClientToScreen(&ptDropPoint);
        qr_pDragImage->DragMove(ptDropPoint);
    }
    CDialog::OnMouseMove(nFlags, point);
}

/*****
 *   Displaying remote task queue and scheduling dialogs
 * *****/
void QueryRetrieve::OnServicesShowRemoteTasks()
{
    qr_DQRCtrlRemote.ShowTaskView();
    UpdateWindow(); // Make sure it redraws the window
}

void QueryRetrieve::OnUpdateServicesShowRemoteTasks(CCmdUI* pCmdUI)
{
    pCmdUI->Enable(qr_DQRCtrlRemote.HasTaskQueue());
}

void QueryRetrieve::OnServicesTaskScheduling()
{
    // Switch scheduling prompt
    qr_DQRCtrlRemote.SetTaskSchedulerPrompt(
        !qr_DQRCtrlRemote.GetTaskSchedulerPrompt());
    qr_UpdateMenu = true;
}

void QueryRetrieve::OnUpdateServicesTaskScheduling(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(qr_DQRCtrlRemote.GetTaskSchedulerPrompt());
}

```

```
// Server.h: interface for the Server class.
//
/////////////////////////////////////////////////////////////////

#ifndef _SERVER_H_INCLUDED_
#define _SERVER_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Server
{
public:
    bool RunServer(ApplicationEntity* a, bool multithread);

    Server(DICOMDatabase* db, DICOMViewLog* log);
    virtual ~Server();

private:
    UINT16      srv_CancelledID;
    DICOMViewLog* srv_pLog;
    DICOMDatabase* srv_pDataBase;
    struct Thread
    {
        char      tr_LocAET[20], tr_RemAET[20];
        UINT16*    tr_ptrCancelledID;
        int        tr_socket, tr_ServPort;
        static int tr_Count;
        DICOMViewLog* tr_pLog;
        DICOMDatabase* tr_pDataBase;

        void      StartThread(char* locAET, char* remAET, bool multithread);
        static void StartFunc(void *pThread);
        bool      RunThread();
        Thread (DICOMDatabase* db, DICOMViewLog* log, int socketfd);
        ~Thread() { tr_Count--; };
    };

    bool RunServer(char* port, char* locAET, char* remAET, bool multithread);
};

#endif // !defined(_SERVER_H_INCLUDED_)
```

1

```

// Start server thread. This function will delete the thread "trd"
// after completion. Never use "thr" any more !
trd->StartThread(locAET, remAET, multithread);
MSocket.Socketfd = 0;
MSocket.Connected = FALSE;
}
sprintf(info, "Server failed to accept more data at port %s", sport);
srv_pLog->Load(info);
return false; // should never return except on error
}

bool Server::RunServer(ApplicationEntity* a, bool multithread)
{
    if(!a) return false;
    return RunServer(a->GetPortServerString(), a->ae_partnerTitle,
        a->ae_Title, multithread);
}

/*****
 *
 * Start a thread, if possible
 *
 *****/
void Server::Thread::StartThread(char* locAET, char* remAET, bool multithread)
{
    strcpy(tr_LocAET, locAET);
    strcpy(tr_RemAET, remAET);
    if(multithread)
    {
        if(_beginthread(StartFunc, 100000, this) == -1)
        {
            try
            {
                tr_pLog->Load("Server WARNING: cannot start a new thread");
                RunThread(); // just try to run
            }
            catch (...) {}
        }
    }
    else RunThread();
}

/*****
 *
 * Run a thread, wrapped in C syntax
 *
 *****/
void Server::Thread::StartFunc(void *pThread)
{
    if(!pThread) return; // cannot be NULL
    ((Thread*)pThread)->RunThread();
    _endthread();
}

/*****
 *
 * Run and DELETES a thread
 *
 *****/
bool Server::Thread::RunThread()
{
    if(!tr_pDataBase || ! tr_pLog)
    {
        delete this;
        return false;
    }
    char info[64];
    PDU_Service PDU;
    DICOMCommandObject DCO;
    UID uid;
    SCPProvider SCP(*tr_pLog, *tr_pDataBase);

    // Configure PDU
    PDU.ClearAbstractSyntaxes();
    PDU.SetLocalAddress((BYTE*)tr_LocAET);

```

```

PDU.SetRemoteAddress((BYE tr_RemAET);
uid.Set("1.2.840.10008.3.1"); PDU.SetApplicationContext(uid);
PDU.AddStandardStoreAbstractSyntaxes();

// NOTE: PDU.Multiplex(filedes). This call is saying, here's a just
// connected socket, now go through the regular PDU association, and
// return to me a connected DICOM link.

if(PDU.Multiplex(tr_socket))
{
    while(TRUE)
    {
        tr_pLog->ReportTime();
        DCO.Reset();
        if(!PDU.Read(&DCO))
        {
            PDU.Close();
            delete this;
            return false;
        }
        sprintf(info, "Server object received at port %d:", tr_ServPort);
        tr_pLog->Load(info);
        tr_pLog->Load(DCO);
        Beep(300,100);
        if(!SCP.IdentifyAndProcess(PDU, DCO, tr_ptrCancelledID)) break;
    } // while
} // if
else
{
    switch (((AAssociateRJ) PDU).Reason)
    {
    case 3:
        tr_pLog->Load("Server ERROR: Rejected remote AE address");
        break;
    case 7:
        tr_pLog->Load("Server ERROR: Rejected local AE address");
        break;
    case 2:
        tr_pLog->Load("Server ERROR: Rejected proposed Application Context");
        break;
    default:
        tr_pLog->Load("Server ERROR: Reason unknown");
    }
    PDU.Close();
    delete this;
    return false;
}
// We get here on success
PDU.Close();
delete this;
return true;
}

```

```

// AccurateTimer.h: interface and implementation of the AccurateTimer class.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DCM.h"

#ifdef _DEBUG
#ifdef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif
#endif

class AccurateTimer
{
private :
    int Initialized;
    __int64 Frequency;
    __int64 BeginTime;
public :
    AccurateTimer() // constructor
    {
        // get the frequency of the counter
        Initialized = QueryPerformanceFrequency( (LARGE_INTEGER *)&Frequency );
    }

    BOOL Begin() // start timing
    {
        if( ! Initialized ) return 0; // error - couldn't get frequency
        // get the starting counter value
        return QueryPerformanceCounter( (LARGE_INTEGER *)&BeginTime );
    }

    double End() // stop timing and get elapsed time in seconds
    {
        if( ! Initialized ) return 0.0; // error - couldn't get frequency
        // get the ending counter value
        __int64 endtime;
        QueryPerformanceCounter( (LARGE_INTEGER *)&endtime );
        // determine the elapsed counts
        __int64 elapsed = endtime - BeginTime;
        // convert counts to time in seconds and return it
        return (double)elapsed / (double)Frequency;
    }

    void EndReport() // stop timing and get elapsed time in seconds
    {
        double t=End();
        CString time; time.Format("Time=%lf seconds",t);
        AfxMessageBox(time);
    }

    BOOL Available() // returns true if the perf counter is available
    { return Initialized; }

    __int64 GetFreq() // return perf counter frequency as large int
    { return Frequency; }
};

```

```
// Angle.h: interface for the Angle class.
//
/////////////////////////////////////////////////////////////////
#ifndef AFX_ANGLE_H__0AD1D023_EE35_11D2_963D_00105A21774F__INCLUDED_
#define AFX_ANGLE_H__0AD1D023_EE35_11D2_963D_00105A21774F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////
// Angle command target

class Angle
{
public:
    bool a_undo, a_active;

    void UpdatePopupMenu(CMenu* pop);
    void SetScale(int code);
    void Redraw(CDC* pDC, CPoint &p, bool initialize=false);
    void Draw(CDC* pDC);
    CString toString();
    Angle();
    virtual ~Angle();

private:
    double a_angle, a_scale_coeff;
    CPoint a_p1, a_p2, a_p3;
    CString a_scale;

    void Clean();
    void GetAngle();

/////////////////////////////////////////////////////////////////
#endif // !defined(AFX_ANGLE_H__0AD1D023_EE35_11D2_963D_00105A21774F__INCLUDED_)
```



```

// Angle.cpp: implementation of the Angle class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "DCM.h"
#include "Angle.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

Angle::Angle()
{
    Clean();
}

Angle::~Angle()
{
}

/*****
 * Report current angle in the status bar
 *****/
CString Angle::toString()
{
    CString info;
    info.Format("%.2lf ", a_angle);
    return (info+a_scale);
}

/*****
 * Compute current angle
 *****/
void Angle::GetAngle()
{
    a_angle=0.0;
    double x=_hypot(a_p1.x-a_p2.x,a_p1.y-a_p2.y);    if(x<=0.1) return;
    double y=_hypot(a_p3.x-a_p2.x,a_p3.y-a_p2.y);    if(y<=0.1) return;
    double z=_hypot(a_p1.x-a_p3.x,a_p1.y-a_p3.y);    if(z<=0.1) return;
    double cosa=(x*x+y*y-z*z)/(2*x*y);
    if(cosa>0.999) a_angle=3.1415926;
    else if(cosa<-0.999) a_angle=3.1415926;
    else a_angle=acos(cosa);
    a_angle *= a_scale_coeff;
    return;
}

/*****
 * Reset all Angle parameters
 *****/
void Angle::Clean()
{
    a_p1=CPoint(20,30);
    a_p2=CPoint(50,50);
    a_p3=CPoint(20,70);
    a_scale=CString("degrees");
    a_scale_coeff=180/3.1415926;
    GetAngle();
}

```

```

    a_active=false;
    a_undo=false;
}

/*****
*
*   Draw the Angle
*
*****/
void Angle::Draw(CDC *pDC)
{
    int dmode=SetROP2(pDC->m_hDC, R2_NOT);
    CPen* old_pen = pDC->SelectObject(&theApp.app_Pen);

    // Draw Angle lines
    pDC->MoveTo(a_p1); pDC->LineTo(a_p2);
    pDC->MoveTo(a_p2); pDC->LineTo(a_p3);
    // Circle the Angle points
    pDC->Arc(CRect(a_p1.x-6,a_p1.y-6,a_p1.x+6,a_p1.y+6), a_p1, a_p1);
    pDC->Arc(CRect(a_p2.x-6,a_p2.y-6,a_p2.x+6,a_p2.y+6), a_p2, a_p2);
    pDC->Arc(CRect(a_p2.x-4,a_p2.y-4,a_p2.x+4,a_p2.y+4), a_p2, a_p2);
    pDC->Arc(CRect(a_p3.x-6,a_p3.y-6,a_p3.x+6,a_p3.y+6), a_p3, a_p3);

    SetROP2(pDC->m_hDC, dmode);
    pDC->SelectObject(old_pen);
}

/*****
*
*   Update the Angle
*
*****/
void Angle::Redraw(CDC *pDC, CPoint &p, bool initialize)
{
    if(!a_active) return;

    // Initialize angle points
    if(initialize)
    {
        if(a_undo) Draw(pDC);
        int px=(p.x<=1)-10; int py=(p.y<=1)-10;
        if(a_p1.x>px) a_p1.x=px;    if(a_p1.y>py) a_p1.y=py;
        if(a_p3.x>px) a_p3.x=px;    if(a_p3.y>py) a_p3.y=py;
        a_p2=p;
        Draw(pDC); // new Angle
        GetAngle();
        a_undo=true;
        return;
    }

    // Find which Angle vertex is closer
    double x1=_hypot(p.x-a_p1.x,p.y-a_p1.y);
    double x2=_hypot(p.x-a_p2.x,p.y-a_p2.y);
    double x3=_hypot(p.x-a_p3.x,p.y-a_p3.y);

    // Update the nearest vertex
    if(x1<=x2 && x1<=x3)
    {
        if(x2<=10 || x3<=10) return; // avoid degraded Angle
        if(a_undo) Draw(pDC);
        a_p1=p;
    }
    else if(x2<=x1 && x2<=x3)
    {
        if(x1<=10 || x3<=10) return; // avoid degraded Angle
        if(a_undo) Draw(pDC);
        a_p2=p;
    }
    else
    {
        if(x1<=10 || x2<=10) return; // avoid degraded Angle
    }
}

```

```

        if(a_undo) Draw(pDC)
        a_p3=p;
    }

    Draw(pDC); // new Angle
    GetAngle();
    a_undo=true;
}

/*****
*
*   Update Angle scale
*
*****/
void Angle::SetScale(int code)
{
    switch(code)
    {
    case 1:
        a_scale="degrees";
        a_scale_coeff=180/3.1415926;
        break;
    case 2:
        a_scale="radians";
        a_scale_coeff=1.0;
        break;
    case 3:
        a_scale="percent";
        a_scale_coeff=50.0/3.1415926;
        break;
    }
}

/*****
*
*   Update Angle pop-up menu
*
*****/
void Angle::UpdatePopupMenu(CMenu *pop)
{
    GetAngle();
    if(a_scale=="degrees") pop->CheckMenuItem(ID_ANGLE_DEGREES,MF_CHECKED );
    else if (a_scale=="radians") pop->CheckMenuItem(ID_ANGLE_RADIANS,MF_CHECKED );
    else pop->CheckMenuItem(ID_ANGLE_PERCENT,MF_CHECKED );
    pop->ModifyMenu(ID_ANGLE_VALUE,MF_BYCOMMAND,ID_ANGLE_VALUE,toString());
}

```

```

// CustomFileDialog.h : header file
//

/////////////////////////////////////////////////////////////////
// CCustomFileDialog dialog
#include "resource.h"

#define CUSTOM_FILEOPENORD          1538
#define CUSTOM_MULTIFILEOPENORD    1539

class CCustomFileDialog : public CFileDialog
{
    DECLARE_DYNAMIC(CCustomFileDialog)
public:
    TCHAR            m_szBigBuffer[1000];
    static CString   szCustomDefFilter;
    static CString   szCustomDefExt;
    static CString   szCustomDefFileName;
    static CString   szCustomTitle;
    CStringList      m_listDisplayNames;

    void             SetTitle(CString title);
    int              GetSelectedCount() { return m_listDisplayNames.GetCount(); }
    CString          GetSelectedAt(UINT index);
    CCustomFileDialog(BOOL bOpenFileDialog = TRUE, // TRUE for FileOpen, FALSE for FileSaveAs
        DWORD dwFlags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT | OFN_NODEREFERENCELINKS | OFN_EXPL
ORER |
        OFN_ENABLETEMPLATE | OFN_ALLOWMULTISELECT | OFN_FILEMUSTEXIST,
        LPCTSTR lpszFilter = CCustomFileDialog::szCustomDefFilter,
        LPCTSTR lpszDefExt = CCustomFileDialog::szCustomDefExt,
        LPCTSTR lpszFileName = CCustomFileDialog::szCustomDefFileName,
        CWnd* pParentWnd = NULL);

    Dialog Data
    //{AFX_DATA(CCustomFileDialog)
    enum { IDD = IDD_CUSTOM_FILE_DIALOG };
    BOOL            m_bMulti;
    BOOL            m_SelectSubdirectories;
    //}AFX_DATA

    Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CCustomFileDialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}AFX_VIRTUAL
    virtual BOOL OnFileNameOK();

    // Implementation
protected:
    BOOL ReadListViewNames();    // protected -> not callable without dialog up

    //{AFX_MSG(CCustomFileDialog)
    afx_msg void OnSelectButton();
    afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    afx_msg void OnHelp();
    DECLARE_MESSAGE_MAP()
};

```

```

switch(m_FromIndex)
{
case 0: // now
    m_StartTime = CTime::GetCurrentTime();
    break;
case 1: // in 10 min
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,0,10,0);
    break;
case 2: // in 30 min
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,0,30,0);
    break;
case 3: // in 1 hour
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,1,0,0);
    break;
case 4: // in 2 hours
    m_StartTime = CTime::GetCurrentTime() + CTimeSpan(0,2,0,0);
    break;
case 5: // tomorrow 8:00 am
    tm = CTime::GetCurrentTime() + CTimeSpan(1,0,0,0);
    m_StartTime = CTime(tm.GetYear(), tm.GetMonth(), tm.GetDay(),
                        8,0,0);
    break;
case 6: // specific
    tm = m_StartTime;
    m_StartTime = CTime(m_StartDate.GetYear(), m_StartDate.GetMonth(),
                        m_StartDate.GetDay(), tm.GetHour(),
                        tm.GetMinute(), tm.GetSecond() );
    if(m_StartTime < CTime::GetCurrentTime())
    {
        m_StartTime = CTime::GetCurrentTime();
    }
    break;
}
// 2. Find out end date/time
switch(m_ToIndex)
{
case 0: // undefined
    m_EndTime = CTime(2030,1,1,1,1,1);
    break;
case 1: // in 10 min
    m_EndTime = m_StartTime + CTimeSpan(0,0,10,0);
    break;
case 2: // in 30 min
    m_EndTime = m_StartTime + CTimeSpan(0,0,30,0);
    break;
case 3: // in 1 hour
    m_EndTime = m_StartTime + CTimeSpan(0,1,0,0);
    break;
case 4: // in 2 hours
    m_EndTime = m_StartTime + CTimeSpan(0,2,0,0);
    break;
case 5: // tomorrow 8:00 am
    tm = CTime::GetCurrentTime() + CTimeSpan(1,0,0,0);
    m_EndTime = CTime(tm.GetYear(), tm.GetMonth(), tm.GetDay(),
                        8,0,0);
    if(m_EndTime <= m_StartTime )
    {
        m_EndTime = m_StartTime + CTimeSpan(0,0,30,0);
    }
    break;
case 6: // specific
    tm = m_EndTime;
    m_EndTime = CTime(m_EndDate.GetYear(), m_EndDate.GetMonth(),
                        m_EndDate.GetDay(), tm.GetHour(),
                        tm.GetMinute(), tm.GetSecond() );
    if(m_EndTime <= m_StartTime )
    {
        m_EndTime = m_StartTime + CTimeSpan(0,0,30,0);
    }
    break;
}

// 3. Set task schedule
DateTime      dstart, dend;

```

```

dstart.SetDateTime(m_StartTime.GetYear(), m_StartTime.GetMonth()-1,
    m_StartTime.GetDay(), m_StartTime.GetHour(), m_StartTime.GetMinute(),
    m_StartTime.GetSecond());
if (m_FromIndex>0)
{
    dend.SetDateTime(m_EndTime.GetYear(), m_EndTime.GetMonth()-1,
        m_EndTime.GetDay()-1, m_EndTime.GetHour(), m_EndTime.GetMinute(),
        m_EndTime.GetSecond());
}
t.ScheduleTask(dstart,dend);

// 4. Set number of execution attempts
t.SetExec(m_Attempts);
}

```

```

/*****
 *
 *   Display schedule dialog and schedule the task
 *
 *****/
void DQRTaskSchedule::RunScheduler(DQRTask &t, bool prompt)
{
    if(prompt)
    {
        if(DoModal() != IDOK)    return;
    }
    ScheduleTask(t);
}

```

```

// ODBC.h: interface for the ODBC class.
//
/////////////////////////////////////////////////////////////////
#ifndef AFX_ODBC_H_INCLUDED_
#define AFX_ODBC_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////
//
// ODBCTableSet recordset - abstract parent for all ODBCRecordSets
//
/////////////////////////////////////////////////////////////////
class ODBCTableSet : public CRecordset
{
public:
    void WriteIntoDICOMObject(DICOMObject& dob,
                              DICOMObject* dob_mask=NULL);
    virtual void WriteIntoDICOMRecord(DICOMRecord& dr)=0;
    virtual void SetFindFilter(DICOMRecord& dr)=0;
    bool FetchNextRecord();
    bool AddOrUpdate(DICOMRecord &dr);
    virtual CString GetFilename() { return CString(""); };
    ODBCTableSet(CDatabase* pDatabase = NULL);
    ~ODBCTableSet();
    DECLARE_DYNAMIC(ODBCTableSet)
}

// Implementation
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    void AndFilter(CString fName, DateTimeSegment& dValue,
                  const BYTE dFormat);
    void AndFilter(CString fName, char* sValue);
    virtual void ClearSet()=0;
    virtual void UpdateFromDICOMRecord(DICOMRecord &dr)=0;
    virtual bool SetFromDICOMRecord(DICOMRecord &dr)=0;
    CString GetDefaultConnect();
    virtual CString GetDefaultQuery(DICOMRecord& dr)=0;
}

/////////////////////////////////////////////////////////////////
// ODBC4Set recordset
//
/////////////////////////////////////////////////////////////////

class ODBC4Set : public ODBCTableSet
{
public:
    void WriteIntoDICOMRecord(DICOMRecord& dr);
    bool HasAliasRecords(DICOMRecord& dr);
    CString GetFilename() { return m_Filename; };
    ODBC4Set(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBC4Set)

// Field/Param Data
//{{AFX_FIELD(ODBC4Set, CRecordset)
    CString m_SeriesInstUID;
    CString m_SOPInstUID;
    CString m_ImageNum;
    CString m_Filename;
    CString m_PatientID;
    CString m_PatientName;
    double m_PBirthTime;
    long m_PBirthDate;
    CString m_StudyInstUID;
    CString m_SeriesInstUID2;
    CString m_Modality;

```

```

CString m_SeriesNum;
CString m_PatientID2;
CString m_StudyInstUID2;
CString m_StudyID;
CString m_AccessionNumber;
double m_StudyTime;
long m_StudyDate;
CString m_StudyImagesNum;
//}}AFX_FIELD

```

```
// Overrides
```

```

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBC4Set)
public:
virtual CString GetDefaultSQL(); // Default SQL for Recordset
virtual void DoFieldExchange(CFieldExchange* pFX); // RFX support
//}}AFX_VIRTUAL

```

```
// Implementation
```

```
#ifdef _DEBUG
```

```
#endif
```

```
private:
```

```

void ClearSet();
void SetFindFilter(DICOMRecord& dr);
void UpdateFromDICOMRecord(DICOMRecord &dr) { ; } ; // not used
bool SetFromDICOMRecord(DICOMRecord &dr) { return true; } ; // not used
CString GetDefaultQuery(DICOMRecord& dr);

```

```
Class ODBCDatabase : public DICOMDatabase
```

```
public:
```

```

static const CString db_DBName;

void RemoveAllRecords();
void StopSearch(void* pTR);
void DisplayRecords(Array<DICOMRecord> &a, bool from_local);
bool InitializeDataBase(char* directory,
                        void (*disp)(Array<DICOMRecord>&, bool));
bool DBAdd(DICOMRecord& dr);
bool DBAdd(DICOMObject* dob, PDU_Service* pdu);
bool GetFromLocal(DICOMDataObject& ddo_mask);
bool SetRecordCount(int c) { return true; }; // unused
BYTE MatchNext(void* pTR, DICOMObject &dob_found,
                DICOMObject *dob_mask);
BYTE RetrieveNext(void* pTR, DICOMObject& dob_found);
int DBRemove(DICOMRecord& dr_mask);
int GetRecordCount() { return 1; }; // unused
CDatabase* GetCDatabasePtr() { return &db_ODBCdb; };
ODBCDatabase();
virtual ~ODBCDatabase();

```

```
protected:
```

```

void* StartSearch(DICOMRecord& dr_mask, const BYTE how);
bool DBAddDirectoryContents(char* directory, bool copy_files,
                           bool include_subdirectories=true);
int DBRemoveDirectoryContents(char *directory, bool use_filenames,
                              bool include_subdirectories=true);

```

```
private:
```

```

bool db_IsODBC;
CDatabase db_ODBCdb;
CCriticalSection
db_DBAddDDO_CriticalSection, db_DisplayRecords_CriticalSection;

bool RemoveUnique(CString& pKey, CString& stKey,
                  CString& serKey, CString& imKey);
bool Connect();

```

```
};
```

```

////////////////////////////////////
//

```



```

// ODBCPatientSet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCPatientSet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord& dr);
    void        SetFindFilter(DICOMRecord& dr);
    ODBCPatientSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCPatientSet)

// Field/Param Data
//{{AFX_FIELD(ODBCPatientSet, CRecordset)
    CString m_PatientID;
    CString m_PatientName;
    double m_PBirthTime;
    long m_PBirthDate;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCPatientSet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX);  // RFX support
//}}AFX_VIRTUAL

// Implementation
private:
    void        ClearSet();
    void        UpdateFromDICOMRecord(DICOMRecord& dr);
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord& dr);
};

/////////////////////////////////////////////////////////////////
// ODBCStudySet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCStudySet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord& dr);
    void        SetFindFilter(DICOMRecord &dr);
    ODBCStudySet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCStudySet)

// Field/Param Data
//{{AFX_FIELD(ODBCStudySet, CRecordset)
    CString m_PatientID;
    CString m_StudyInstUID;
    CString m_StudyID;
    CString m_AccessionNumber;
    double m_StudyTime;
    long m_StudyDate;
    CString m_StudyImagesNum;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCStudySet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX);  // RFX support
//}}AFX_VIRTUAL
private:
    void        UpdateFromDICOMRecord(DICOMRecord &dr);
    void        ClearSet();
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord& dr);
};

```

```

};

/////////////////////////////////////////////////////////////////
//
// ODBCSeriesSet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCSeriesSet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord &dr);
    void        SetFindFilter(DICOMRecord &dr);
    ODBCSeriesSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCSeriesSet)

// Field/Param Data
//{{AFX_FIELD(ODBCSeriesSet, CRecordset)
CString m_StudyInstUID;
CString m_SeriesInstUID;
CString m_Modality;
CString m_SeriesNum;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCSeriesSet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX); // RFX support
//}}AFX_VIRTUAL
private:
    void        UpdateFromDICOMRecord(DICOMRecord &dr);
    void        ClearSet();
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord &dr);
}

/////////////////////////////////////////////////////////////////
// ODBCImageSet recordset
//
/////////////////////////////////////////////////////////////////
class ODBCImageSet : public ODBCTableSet
{
public:
    void        WriteIntoDICOMRecord(DICOMRecord &dr);
    void        SetFindFilter(DICOMRecord &dr);
    CString     GetFilename() { return m_Filename; };
    ODBCImageSet(CDatabase* pDatabase = NULL);
    DECLARE_DYNAMIC(ODBCImageSet)

// Field/Param Data
//{{AFX_FIELD(ODBCImageSet, CRecordset)
CString m_SeriesInstUID;
CString m_SOPInstUID;
CString m_ImageNum;
CString m_Filename;
//}}AFX_FIELD

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(ODBCImageSet)
public:
    virtual CString GetDefaultSQL();    // Default SQL for Recordset
    virtual void DoFieldExchange(CFieldExchange* pFX); // RFX support
//}}AFX_VIRTUAL
private:
    void        UpdateFromDICOMRecord(DICOMRecord &dr);
    void        ClearSet();
    bool        SetFromDICOMRecord(DICOMRecord &dr);
    CString     GetDefaultQuery(DICOMRecord &dr);
};

```

```
#endif // !defined(AFX_ODBC_...CLUDED_)
```

```

////////////////////////////////////
// ODBCDatabase Class
// This class is derived from DICOMDatabase to support
// ODBC data sources
////////////////////////////////////
#include "stdafx.h"
#include <odbcinst.h>
#include "ODBC.h"
#include "../DCM.h"

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////
const CString ODBCDatabase::db_DBName = "DCMDBase";

ODBCDatabase::ODBCDatabase()
{
    db_IsODBC=false;
}

ODBCDatabase::~ODBCDatabase()
{
    // Close database connection
    TRY { if(db_ODBCdb.IsOpen()) db_ODBCdb.Close(); }
    CATCH(CException, e) { ; }
    END_CATCH;
}

/*****
*
* Add records
*****/
bool ODBCDatabase::DBAdd(DICOMRecord &dr)
{
    if(!db_IsODBC) return DICOMDatabase::DBAdd(dr);
    if(!dr.HasUniquePrimaryKeys()) return false;
    // Check for consistency
    ODBC4Set set;
    if(set.HasAliasRecords(dr)) return false; // violates DB tree structure
    // Insert
    ODBCPatientSet ps;
    if(!ps.AddOrUpdate(dr)) return false;
    ODBCStudySet sts;
    if(!sts.AddOrUpdate(dr)) return false;
    OBCSeriesSet srs;
    if(!srs.AddOrUpdate(dr)) return false;
    ODBCImageSet ims;
    if(!ims.AddOrUpdate(dr)) return false;
    db_MostRecentRecord=dr;
    return true;
}

/*****
*
* Add objects, thread-safe
*****/
bool ODBCDatabase::DBAdd(DICOMObject *dob, PDU_Service *pdu)
{
    bool success = false;
    CSingleLock singleLock(&db_DBAddDDO_CriticalSection);
    singleLock.Lock(3000); // attempt to lock the shared resource
    if (singleLock.IsLocked()) // resource has been locked
    {
        success = DICOMDatabase::DBAdd(dob, pdu);
    }
    singleLock.Unlock();
    return success;
}

/*****
*
* Display records, thread-safe
*****/

```

```

*****
void ODBCDatabase::DisplayRecords(Array<DICOMRecord> &a, bool from_local)
{
    CSingleLock singleLock(&db_DisplayRecords_CriticalSection);
    singleLock.Lock(10000); // attempt to lock the shared resource
    if (singleLock.IsLocked()) // resource has been locked
    {
        DICOMDatabase::DisplayRecords(a, from_local);
    }
    singleLock.Unlock();
}
/*****
*
*   Add files and directories
*
*****/
bool ODBCDatabase::DBAddDirectoryContents(char *directory, bool copy_files,
                                          bool include_subdirectories/*=true*/)
{
    CString info("Adding directory ");
    info += CString(directory);

    // Process individual file
    if(DICOMDatabase::DBAddDirectoryContents(directory, copy_files,
                                              include_subdirectories)) return true;
    // Processing Windows directory
    WIN32_FIND_DATA wf;
    // Add '\\' to the end of directory string, if needed
    char dir[MAX_PATH];
    char clast = directory[strlen(directory)-1];
    if(clast=='/' || clast=='\\') sprintf(dir,"%s",directory);
    else                          sprintf(dir,"%s\\",directory);
    // Set wildcard search mask
    char nf[MAX_PATH];
    sprintf(nf,"%s*",dir);
    // Search
    char fullname[MAX_PATH];
    HANDLE hf=FindFirstFile(nf,&wf);
    if(hf==INVALID_HANDLE_VALUE) return false;
    do
    {
        if(strcmp(wf.cFileName,".")==0) continue;
        if(strcmp(wf.cFileName,"..")==0) continue;
        sprintf(fullname,"%s%s",dir,wf.cFileName);
        if(wf.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        {
            if(include_subdirectories)
            {
                DBAddDirectoryContents(fullname, copy_files);
            }
        }
        else
        {
            DICOMDatabase::DBAdd(fullname, copy_files);
        }
    }
    while (FindNextFile(hf,&wf));
    FindClose(hf);
    return true;
}
/*****
*
*   Remove files and directories
*
*****/
int ODBCDatabase::DBRemoveDirectoryContents(char *directory,
                                             bool use_filenames, bool include_subdirectories/*=true*/)
{
    CString info("Removing directory ");
    info += CString(directory);

    // Process individual file
    if(DICOMDatabase::DBRemoveDirectoryContents(directory, use_filenames,
                                              include_subdirectories)) return true;
}

```

```

// Processing Windows directory
WIN32_FIND_DATA wf;
// Add '\\\' to the end of directory string, if needed
char dir[MAX_PATH];
char clast = directory[strlen(directory)-1];
if(clast=='/' || clast=='\\')    sprintf(dir,"%s",directory);
else                            sprintf(dir,"%s\\",directory);
// Set wildcard search mask
char nf[MAX_PATH];
sprintf(nf,"%s*",dir);
// Search
char fullname[MAX_PATH];
HANDLE hf=FindFirstFile(nf,&wf);
if(hf==INVALID_HANDLE_VALUE)    return false;
do
{
    if(strcmp(wf.cFileName,".")==0)    continue;
    if(strcmp(wf.cFileName,"..")==0)    continue;
    sprintf(fullname,"%s%s",dir,wf.cFileName);
    if(wf.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
    {
        if(include_subdirectories)
        {
            DBRemoveDirectoryContents(fullname, use_filenames);
        }
    }
    else
    {
        DICOMDatabase::DBRemove(fullname, use_filenames);
    }
}
while (FindNextFile(hf,&wf));
FindClose(hf);
return true;

```

```

*****
Remove all records from the database

```

```

*****/
bool ODBCDatabase::RemoveAllRecords()
{
    if(AfxMessageBox("Are you sure you want to remove all database records ?",
        MB_YESNO) == IDNO)    return;
    DICOMRecord dr;
    int n = DBRemove(dr);
    if(n>0)
    {
        CString info;
        info.Format("%d records were removed", n);
        AfxMessageBox(info, MB_ICONINFORMATION);
    }
    else AfxMessageBox("No records found", MB_ICONINFORMATION);
}

```

```

/*****
*
*   Open local records
*
*****/
bool ODBCDatabase::GetFromLocal(DICOMDataObject &ddo_mask)
{
    if(!db_IsODBC)    return DICOMDatabase::GetFromLocal(ddo_mask);
    ODBCTableSet* pTR=NULL;
    pTR = (ODBCTableSet*)DICOMDatabase::StartSearch(ddo_mask,
        RetrieveHierarchical);
    if(!pTR)    return false;
    Array<DICOMRecord> found;
    CString    fname;
    do
    {
        fname = pTR->GetFilename();
        if(fname=="")    continue;
    }
}

```

```

        pTR->WriteIntoDICOMRecord(db_MostRecentRecord);
        found.Add(db_MostRecentRecord);
    }
    while(pTR->FetchNextRecord());
    StopSearch(pTR);
    DisplayRecords(found, true);
    return true;
}

/*****
 *
 *   Initialize database
 *
 *****/
bool ODBCDatabase::InitializeDataBase(char *directory,
                                     void (__cdecl *disp)(Array<DICOMRecord>&, bool))
{
    if(!DICOMDatabase::InitializeDataBase(directory, disp)) return false;
    // Try to open ODBC connection
    if(Connect()) return true;
    // Try to find the local DB file
    CString failedODBC = CString("DCM failed to setup ODBC database.\n")+
        CString("File-based database will be used instead");
    CString dbfilenew = CString(directory);
    int n = dbfilenew.Find("\\Applic",0); if(n>0) dbfilenew=dbfilenew.Left(n);
    CString dbfileold;
    dbfileold.Format("%s\\%s.mdb",dbfilenew,db_DBName);
    dbfilenew.Format("%s\\%s.mdb",directory,db_DBName);
    if(GetFileAttributes(dbfilenew)==-1) // no such file
    {
        if(CopyFile(dbfileold, dbfilenew, FALSE) == FALSE)
        {
            CString nofile;
            nofile.Format("Cannot find ODBC database file %s.\n",dbfileold);
            AfxMessageBox(nofile+failedODBC, MB_ICONINFORMATION|MB_OK);
            return true;
        }
    }
    // Try to reinstall the ODBC
    CString driver = "Microsoft Access Driver (*.mdb)";
    CString config;
    config.Format("DSN=%s; DESCRIPTION=DCM Application Database;"
        "DBQ=%s;",db_DBName, dbfilenew);

    if(::SQLConfigDataSource(NULL,ODBC_ADD_DSN,
        driver,config) == FALSE)
    {
        AfxMessageBox(failedODBC, MB_ICONINFORMATION|MB_OK);
    }
    else
    {
        if(!Connect()) AfxMessageBox(failedODBC, MB_ICONINFORMATION|MB_OK);
    }
    return true;
}

/*****
 *
 *   Remove records. Returns the number of deleted records
 *
 *****/
int ODBCDatabase::DBRemove(DICOMRecord &dr_mask)
{
    if(!db_IsODBC) return DICOMDatabase::DBRemove(dr_mask);
    ODBC4Set* pTR=NULL;
    pTR = (ODBC4Set*)StartSearch(dr_mask, RetrieveRelational);
    if(!pTR) return false;
    int removed=0;
    do
    {
        RemoveUnique(pTR->m_PatientID, pTR->m_StudyInstUID,
            pTR->m_SeriesInstUID, pTR->m_SOPInstUID);
        removed++;
    }
}

```

```

while(pTR->FetchNextRecord());
StopSearch(pTR);
return removed;
}

/*****
*
*   Search records
*
*****/
void* ODBCDatabase::StartSearch(DICOMRecord &dr_mask, const BYTE how)
{
    if(!db_IsODBC) return DICOMDatabase::StartSearch(dr_mask, how);
    // Initialize appropriate class pointer for match/retrieve
    ODBCTableSet* pTR = NULL;
    TRY
    {
        if(how==MatchRelational || how==RetrieveRelational)
        {
            ODBC4Set* pSet = new ODBC4Set();
            pTR = pSet;
        }
        else if(how==MatchHierarchical)
        {
            BYTE lev = dr_mask.FindQLevel();
            if(lev==DICOMRecord::LevelInvalid)
            {
                ODBCPatientSet* pSet = new ODBCPatientSet();
                pTR = pSet;
            }
            else if(lev==DICOMRecord::LevelPatient)
            {
                ODBCStudySet* pSet = new ODBCStudySet();
                pTR = pSet;
            }
            else if(lev==DICOMRecord::LevelStudy)
            {
                ODBCSeriesSet* pSet = new ODBCSeriesSet();
                pTR = pSet;
            }
            else // lev==LevelSeries or lev==LevelImage
            {
                ODBCImageSet* pSet = new ODBCImageSet();
                pTR = pSet;
            }
        }
        else if(how==RetrieveHierarchical)
        {
            if(!::IsUniqueString(dr_mask.GetSOPInstUID()))
            {
                return ODBCDatabase::StartSearch(dr_mask, RetrieveRelational);
            }
            ODBCImageSet* pSet = new ODBCImageSet();
            pTR = pSet;
        }
        else return NULL;
        if(!pTR) return NULL;
        pTR->SetFindFilter(dr_mask);
        if(!pTR->Open()) { delete pTR; return NULL; }
        if(pTR->IsEOF()) { pTR->Close(); delete pTR; return NULL; }
        return pTR;
    } // TRY
    CATCH(CException, e)
    {
        if(pTR) delete pTR;
        return NULL;
    }
    END_CATCH;
    if(pTR) delete pTR;
    return NULL;
}

void ODBCDatabase::StopSearch(void* pTR)
{
    if(!db_IsODBC) DICOMDatabase::StopSearch(pTR);
}

```



```

    try { delete (ODBCTableSet*) pTR; }
    catch(...) { }
}

BYTE ODBCDatabase::RetrieveNext(void* pTR, DICOMObject &dob_found)
{
    if(!db_IsODBC) return DICOMDatabase::RetrieveNext(pTR, dob_found);
    if(!pTR) return RecordsEnd;
    ODBCTableSet* pTabRec = (ODBCTableSet*)pTR;
    if(!pTabRec->IsOpen() || pTabRec->IsEOF()) return RecordsEnd;
    bool loaded = dob_found.LoadFromFile((char*)(LPCSTR)(pTabRec->GetFilename()));
    pTabRec->FetchNextRecord();
    if(loaded) return RecordFound;
    else return FindMore;
}

BYTE ODBCDatabase::MatchNext(void* pTR, DICOMObject &dob_found,
                             DICOMObject *dob_mask)
{
    if(!db_IsODBC) return DICOMDatabase::MatchNext(pTR, dob_found, dob_mask);
    if(!pTR) return RecordsEnd;
    ODBCTableSet* pTabRec = (ODBCTableSet*)pTR;
    if(!pTabRec->IsOpen() || pTabRec->IsEOF()) return RecordsEnd;
    pTabRec->WriteIntoDICOMObject(dob_found, dob_mask);
    pTabRec->FetchNextRecord();
    return RecordFound;
}

/*****
 *
 * Connect to the application database
 *
 *****/
bool ODBCDatabase::Connect()
{
    db_IsODBC=false;
    TRY
    {
        if(db_ODBCdb.IsOpen()) db_ODBCdb.Close();
        CString con;
        con.Format("DSN=%s;", db_DBName);
        db_IsODBC = (db_ODBCdb.OpenEx(con, CDatabase::noOdbcDialog)==TRUE);
        return db_IsODBC;
    }
    CATCH(CException, e)
    {
        #ifdef _DEBUG
        e->ReportError();
        #endif
        return db_IsODBC;
    }
    END_CATCH;
    return db_IsODBC;
}

/*****
 *
 * Remove an entry with unique primary keys
 *
 *****/
bool ODBCDatabase::RemoveUnique(CString &pKey, CString &stKey,
                                CString &serKey, CString &imKey)
{
    bool stop;
    TRY
    {
        // Remove at image level
        ODBCImageSet ims;
        ims.m_strFilter.Format("SeriesInstUID='%s'", serKey);
        if(!ims.Open()) return false;
        if(!ims.CanUpdate()) { ims.Close(); return false; }
        stop = false;
        while(!ims.IsEOF())
        {
            if(ims.m_SOPInstUID!=imKey) stop=true;
            else
            {
                if(ims.m_Filename.Find(db_Directory,0)>=0)

```

```

        {
            DeleteFile(ims.m_FileName);
        }
        ims.Delete();
    }
    ims.MoveNext();
}
ims.Close();
if(stop)    return true;

// Remove at series level
ODBCSeriesSet srs;
srs.m_strFilter.Format("StudyInstUID='%s'", stKey);
if(!srs.Open()) return false;
if(!srs.CanUpdate( )) {    srs.Close();    return false;    }
stop = false;
while(!srs.IsEOF())
{
    if(srs.m_SeriesInstUID != serKey)    stop=true;
    else    srs.Delete();
    srs.MoveNext();
}
srs.Close();
if(stop)    return true;

// Remove at study level
ODBCStudySet sts;
sts.m_strFilter.Format("PatientID='%s'", pKey);
if(!sts.Open()) return false;
if(!sts.CanUpdate( )) {    sts.Close();    return false;    }
stop = false;
while(!sts.IsEOF())
{
    if(sts.m_StudyInstUID != stKey) stop=true;
    else    sts.Delete();
    sts.MoveNext();
}
sts.Close();
if(stop)    return true;

// Remove at patient level
ODBCPatientSet ps;
ps.m_strFilter.Format("PatientID='%s'", pKey);
if(!ps.Open())    return false;
if(!ps.CanUpdate( )) {    ps.Close(); return false;    }
while(!ps.IsEOF())
{
    ps.Delete();
    if(!ps.IsEOF()) ps.MoveNext();
}
ps.Close();

return true;
}
CATCH(CException, e)
{
    #ifdef _DEBUG
    e->ReportError();
    #endif
    return false;
}
END_CATCH;
return false;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// ODBCTableSet
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

IMPLEMENT_DYNAMIC(ODBCTableSet, CRecordset)

ODBCTableSet::ODBCTableSet(CDatabase* pdb)
: CRecordset(theApp.app_DataBase.GetCDatabasePtr())

```

```

{
    m_nDefaultType = dynaset;
}
// Smart record-closing destructor
ODBCTableSet::~ODBCTableSet()
{
    TRY { if(IsOpen()) Close(); }
    CATCH(CException, e) { ; }
    END_CATCH;
}

CString ODBCTableSet::GetDefaultConnect()
{
    CString connect;
    connect.Format("ODBC;DSN=%s", theApp.app_DataBase.db_DBName);
    return connect;
}

#ifdef _DEBUG
void ODBCTableSet::AssertValid() const
{
    CRecordset::AssertValid();
}

void ODBCTableSet::Dump(CDumpContext& dc) const
{
    CRecordset::Dump(dc);
}

#endif // _DEBUG
//
// Moving to next record, if possible
//
bool ODBCTableSet::FetchNextRecord()
{
    TRY
    {
        if(!IsOpen()) return false;
        MoveNext();
        if(IsEOF()) { Close(); return false; }
        return true;
    }
    CATCH(CException, e) { return false; }
    END_CATCH;
    return false;
}
//
// Append filter criteria
//
//
void ODBCTableSet::AndFilter(CString fName, char *sValue)
{
    if(::IsEmptyString(sValue)) return;
    CString fValue = ::Trim(sValue);
    if(fValue=="") return;

    CString filter("");
    CString and = (m_strFilter == "" ? "" : " AND");
    if(fName.Find("Filename",0)<0 && fValue.Find('\\',0)>0) // we have a set
    {
        fValue.Replace("\\", "'", "'");
        filter.Format("%s %s IN ('%s') ", and, fName, fValue);
        filter.Replace('*', '%'); filter.Replace('?', '_');
    }
    else if(fValue.Find('*',0)>=0 || fValue.Find('?',0)>=0) // we have wildcards
    {
        if(fValue=="*") return; // anything goes
        filter.Format("%s %s LIKE '%s' ", and, fName, fValue);
        filter.Replace('*', '%'); filter.Replace('?', '_');
    }
    else // plain match
    {

```

```

Beep(500,100);
if(!GetSafeHwnd()) return;
if(m_SearchDialog.DoModal() != IDOK) return;
Beep(500,100);
m_RequestedClientService=find_root;
if(!AfxBeginThread(StartQRClient,this,
    GetThreadPriority(),m_ClientStackSize)) RunClientThread();
}
void DQRControl::FindAll()
{
    if(!m_LocalOnly) m_PreloadData=false; // we preload only on local !
    if(m_PreloadData)
    {
        m_RequestedClientService=find_root;
        if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
            m_ClientStackSize)) RunClientThread();
    }
}
void DQRControl::UpdateAfterFind(bool find_success)
{
    LoadFoundList(!find_success);
    if(!find_success) AfxMessageBox("Cannot find", MB_ICONEXCLAMATION|MB_OK);
    if(find_success && m_DQR.GetFoundCount()>0) // Something's found
    {
        // Set list selection
        if(m_RequestedClientService=find_previous)
        {
            int nsel=m_ResultsListSelection[max(1,m_DQR.GetLevel()-1)];
            if(nsel<0 || nsel>m_DQR.GetFoundCount())
            {
                nsel=0;
                m_ButtonGet.EnableWindow(FALSE);
                m_ButtonMoveTo.EnableWindow(FALSE);
            }
            else
            {
                m_ButtonGet.EnableWindow(nsel>0);
                m_ButtonMoveTo.EnableWindow(nsel>0);
            }
            m_ResultsList.SetItemState(nsel,LVIS_SELECTED | LVIS_FOCUSED , LVIS_SELECTED | LVIS_FOCUSED);
            m_ResultsList.EnsureVisible(nsel,FALSE);
        }
    }
}
*****
C-Get
*****/
void DQRControl::OnButtonGet()
{
    Beep(500,100);
    UpdateData(TRUE);
    int sel;
    if(GetResultsListSelection(sel, m_DOBindex)<2)
    {
        AfxMessageBox("Cannot get unspecified entry");
        return;
    }
    m_RequestedClientService=get_index;
    if(m_UseTaskQueue)
    {
        m_DQR.Get(m_DOBindex, true); // queue
    }
    else
    {
        if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
            m_ClientStackSize)) RunClientThread();
    }
}

```

```

/*****
*
*           C-Move
*
*****/
void DQRControl::OnButtonMoveTo()
{
    Beep(500,100);
    UpdatedData(TRUE);
    int sel;
    if(GetResultsListSelection(sel, m_DOBIndex)<2)
    {
        AfxMessageBox("Cannot move unspecified entry");
        return;
    }
    UINT n=m_MoveArchiveList.GetCurSel();
    if(!m_AEarray ) return;
    strcpy(m_MoveDestinationAE, m_AEarray->Get(n).ae_Title);
    m_RequestedClientService=move_index;
    if(m_UseTaskQueue)
    {
        m_DQR.Move(m_DOBIndex,m_MoveDestinationAE,true);    // queue
    }
    else
    {
        if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
            m_ClientStackSize)) RunClientThread();
    }
}

/*****
*
* Run a CLIENT thread, wrapped in C syntax
*
*****/
UINT DQRControl::StartQRClient(LPVOID ptrDQRControl)
{
    DQRControl* pDQRC=(DQRControl*)ptrDQRControl;
    if(pDQRC==NULL) return (UINT)FALSE; // illegal parameter
    return pDQRC->RunClientThread();
}

/*****
*
* Thread-handling routines
*
*****/
void DQRControl::SetInterruptMode(bool interrupt)
{
    int sw = interrupt ? SW_HIDE : SW_SHOW;
    GetDlgItem(IDC_STATIC_MOVE_TO)->ShowWindow(sw);
    if(m_LocalOnly)
    {
        GetDlgItem(IDC_BUTTON_DELETE)->ShowWindow(sw);
    }

    m_ArchiveList.EnableWindow(!interrupt && !m_LocalOnly);
    m_MoveArchiveList.ShowWindow(sw);
    m_ResultsList.EnableWindow(!interrupt);

    m_ButtonSearch.ShowWindow(sw);
    m_ButtonGet.ShowWindow(sw);
    m_ButtonMoveTo.ShowWindow(sw);

    // Enable animation
    if(interrupt)
    {
        m_AnimNetwork.ShowWindow(SW_SHOW);
        GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_SHOW);
        m_AnimNetwork.Open(IDR_AVI_CONNECT);
    }
    else

```

```

{
    m_AnimNetwork.Close();
    m_AnimNetwork.ShowWindow(SW_HIDE);
    m_ButtonCancel.ShowWindow(SW_HIDE);
    GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText("");
    GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_HIDE);
}

/*****
*
*   Thread-safe UpdateData
*
*****/
BOOL DQRControl::UpdateData(BOOL bSaveAndValidate)
{
    if(m_HWND) return FromHandle(m_HWND)->UpdateData(bSaveAndValidate);
    else return FALSE;
}

/*****
*
*   Run client thread
*
*****/
bool DQRControl::RunClientThread()
{
    //Disable window controls
    SetInterruptMode(true);
    // Lock on thread start
    CSingleLock singleLock(&m_RunClientThread_CritSection);
    singleLock.Lock(3000); // attempt to lock the shared resource

    m_DQR.SetLastMessageID(0);

    // Execute DIMSE service
    bool success=false;
    switch(m_RequestedClientService)
    {
    case echo: // C-Echo
        GetDlgItem(IDC_CONNECTION)->
            SetWindowText("Archive connection: Verifying ...");
        success = m_DQR.Echo();
        if(success) GetDlgItem(IDC_CONNECTION)->
            SetWindowText("Archive connection: Connected");
        else
            GetDlgItem(IDC_CONNECTION)->
                SetWindowText("Archive connection: Cannot connect");
        break;
    case find_root:
        {
            DICOMRecord dr;
            m_SearchDialog.WriteIntoDICOMRecord(dr);
            success=m_DQR.FindRoot(dr);
        }
        UpdateAfterFind(success);
        break;
    case find_previous:
        success=m_DQR.FindPreviousLevel();
        UpdateAfterFind(success);
        break;
    case find_next:
        success=m_DQR.FindNextLevel(m_DOBIndex);
        UpdateAfterFind(success);
        break;
    case find:
        if(m_DQR.Find()) UpdateAfterFind(true);
        else
        {
            CString info;
            info.Format("Several records were removed from the local database\n\n"
                "Some of the records in the query results window\n"
                "may no longer be valid !");
            AfxMessageBox(info, MB_ICONINFORMATION);
        }
        break;
    case get_index:

```

```

        success=m_DQR.Get(m_Index, false); // no queue
        LoadFoundList(false);
        if(!success) AfxMessageBox("Cannot get", MB_ICONEXCLAMATION|MB_OK);
        break;
    case move_index:
        success=m_DQR.Move(m_DOBindex,m_MoveDestinationAE,false); // no queue
        LoadFoundList(false);
        if(!success) AfxMessageBox("Cannot move", MB_ICONEXCLAMATION|MB_OK);
        break;
    }
    m_DQR.SetLastMessageID(0);
    SetInterruptMode(false);

    // Signal thread end
    singleLock.Unlock();
    Beep(700,100);
    return success; // normal termination
}
/*****
*
*   Get thread priority from DICOM message priority
*
*****/
int DQRControl::GetThreadPriority()
{
    if(m_DQR.GetPriority() == ServiceClass::LowPriority)
        return THREAD_PRIORITY_BELOW_NORMAL;
    if(m_DQR.GetPriority() == ServiceClass::HighPriority)
        return THREAD_PRIORITY_ABOVE_NORMAL;
    return THREAD_PRIORITY_NORMAL;
}
/*****
*
*   Determine IP address of the local PC
*
*****/
bool DQRControl::GetLocalIP(CString& sname, BYTE& ip1,BYTE& ip2,
                           BYTE& ip3,BYTE& ip4)
{
    bool success = false;
    WORD wVersionRequested;
    WSADATA wsaData;
    char name[255];
    CString ip;
    PHOSTENT hostinfo;
    wVersionRequested = MAKEWORD( 2, 0 );
    sname="";
    if ( WSASStartup( wVersionRequested, &wsaData ) == 0 )
    {
        if( gethostname ( name, sizeof(name) ) == 0 )
        {
            if((hostinfo = gethostbyname(name)) != NULL)
            {
                ip = inet_ntoa (*(struct in_addr *)*hostinfo->h_addr_list);
                sscanf((char*) (LPCSTR) ip, "%u.%u.%u.%u", &ip1,&ip2,&ip3,&ip4);
                success = true;
                sname = CString(name);
                sname.TrimRight(); sname.TrimLeft();
                sname.MakeUpper();
                if(sname=="") sname="This_PC";
            }
        }
        WSACleanup();
    }
    if(!success) { ip1=127; ip2=0; ip3=0; ip4=1; }
    return success;
}

/*****
*
*   Server threads
*
*****/

```

```

*****
UINT DQRControl::StartQRServer(LPVOID index)
{
    UINT ae_index = (UINT)index;
    ApplicationEntityList *ael = (ApplicationEntityList*)
        (&theApp.app_DataBase.db_AElist);
    if(ae_index >= ael->GetSize())
    {
        AfxEndThread(0);    return 0;
    }
    ael->Get(ae_index).SetPartnerTitle(ael->GetLocalAE().ae_Title);
    Server srv(&theApp.app_DataBase, &theApp.app_ServerLog);
    MarkAEServerStatuses(ael, ae_index, true);
    srv.RunServer(&(ael->Get(ae_index)), false); // Multiple threads for CCancel ?
    MarkAEServerStatuses(ael, ae_index, false);
    AfxEndThread(1);
    return 1;
}

void DQRControl::MarkAEServerStatuses(ApplicationEntityList *ael, UINT ae_index,
    bool status)
{
    if(!ael)    return;
    if(ae_index >= ael->GetSize())    return;
    int nPort = ael->Get(ae_index).ae_PortServer;
    ael->SetServedStatus(nPort, status);
    if(!status)
    {
        CString info;    info.Format("Server at port %d abnormally terminated.\n"
            "Please restart the application.", nPort);
        AfxMessageBox(info, MB_ICONINFORMATION|MB_OK);
    }
}

bool DQRControl::StartAEServer(UINT ae_index)
{
    if(!m_AEarray)    return false;
    if(ae_index >= m_AEarray->GetSize())    return false;
    if(!m_AEarray->Get(ae_index).ae_Served) // Need to start server
    {
        // Launch server thread
        if(AfxBeginThread(StartQRServer, (LPVOID)ae_index,
            THREAD_PRIORITY_BELOW_NORMAL, 10000) == NULL)
        {
            return false;
        }
        else // just wait for the server thread to start
        {
            while(m_AEarray->Get(ae_index).ae_Served != true) Sleep(100);
        }
    }
    return true;
}

bool DQRControl::StartAllServers()
{
    if(!m_AEarray)    return false;
    UINT failedCount=0;
    UINT nAEs = m_AEarray->GetSize();
    for(UINT n=0; n< nAEs; n++)
    {
        ::ShowProgress((100*(n+1))/nAEs, "Initializing archive servers");
        if(!StartAEServer(n))    failedCount ++;
        Sleep(10);
    }
    ::ShowProgress(0,0);
    if(failedCount>0)    AfxMessageBox("Failed to launch server(s) for some AEs");
    return (failedCount==0);
}

/*****

```



```

*
*   Function to be called from m_DQR and ServiceClass
*
*****/
int DQRControl::DQRCtrlCallbackFilter(void *dqrctrl, UINT stepnum, UINT id)
{
    if(!dqrctrl)    return 0;
    DQRControl* d;
    try { d = (DQRControl*)dqrctrl; }
    catch(...) { return 0; }
    CString progress;
    if(stepnum == CallbackObject::m_CBConnectingToAE)
    {
        progress = "Connecting to archive ...";
    }
    else if(stepnum == CallbackObject::m_CBConnectionFailed)
    {
        progress = "Connection failed ...";
        d->m_AnimNetwork.Close();
    }
    else if(stepnum == CallbackObject::m_CBSendingRequest)
    {
        progress = "Sending request ...";
        d->m_AnimNetwork.Close();
        d->m_AnimNetwork.Open(IDR_AVI_SEND);
    }
    else if(stepnum == CallbackObject::m_CBGettingResponse)
    {
        progress = "Getting response ...";
        d->m_AnimNetwork.Close();
        d->m_AnimNetwork.Open(IDR_AVI_RECEIVE);
        d->EnableCancel();
    }
    else if(stepnum == CallbackObject::m_CBResponseReceived)
    {
        progress = "Response received";
        d->m_AnimNetwork.Close();
        d->m_AnimNetwork.ShowWindow(SW_HIDE);
        d->m_ButtonCancel.ShowWindow(SW_HIDE);
    }
    else if(stepnum == CallbackObject::m_CBCancelSent)
    {
        progress = "Cancel request sent";
    }
    else if(stepnum == CallbackObject::m_CBDQRTaskSchedule)
    {
        if(d->m_PromptForTaskScheduler)
        {
            d->m_TaskScheduler.RunScheduler(*((DQRTask*)id), true);
        }
        return 1;
    }
    else {}
    if(progress!="")
    {
        d->GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_SHOW);
        d->GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText(progress);
    }
    return d->m_DQR.DQRCallbackFilter(stepnum,id);
}

/*****
*
*   Drag and drop support
*
*****/
void DQRControl::OnBeginDragListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;
    theApp.app_QueryRetrieve.OnBeginDrag(&(this->m_ResultsList),pNMHDR);
    *pResult = 0;
}

void DQRControl::DropOn(CWnd *win)

```

```

{
    if(!win)        return;
    if(win->GetParent() == AfxGetMainWnd() ||
    (win->GetTopLevelFrame() == AfxGetMainWnd() && win->GetParent() != this &&
    win->GetParentFrame() != this->GetParentFrame()))
    {
        OnButtonGet();
        return;
    }
    int nid = win->GetDlgCtrlID();
    if(nid==IDC_LIST_RESULTS)    // drop on results list
    {
        CWnd* par = win->GetParent();
        if(!par || par == (CWnd*)this)    return;
        DQRControl* destin = NULL;
        try
        {
            destin = (DQRControl*)par;
            if(!destin)    return;
            UINT nAEFrom = m_ArchiveList.GetCurSel();
            UINT nAETO = destin->m_ArchiveList.GetCurSel();
            if(nAEFrom == nAETO)
            {
                AfxMessageBox("You cannot copy to the same archive !");
                return;
            }
            m_MoveArchiveList.SetCurSel(nAETO);
            UpdateData(FALSE);
            OnButtonMoveTo();
        }
        catch(...)    {}
        return;
    }
}

void DQRControl::DropOn(CWnd *wdrag, CWnd *wdrop)
{
    if(wdrag != (CWnd*)(&m_ResultsList))    return;
    DropOn(wdrop);
}

*****
Column sorts
*****
void DQRControl::OnHeaderClicked(NMHDR *pNMHDR, LRESULT *pResult)
{
    HD_NOTIFY *phdn = (HD_NOTIFY *) pNMHDR;
    if( phdn->iButton == 0 && m_ResultsList.GetItemCount()>2)
    {
        // User clicked on header using left mouse button
        if( phdn->iItem == m_nSortColumn )
            m_bSortInIncreasingOrder = !m_bSortInIncreasingOrder;
        else
            m_bSortInIncreasingOrder = true;
        m_nSortColumn = phdn->iItem;
        m_ResultsList.SortItems(CompareItems, (DWORD)this);
    }
    *pResult = 0;
}

int CALLBACK DQRControl::CompareItems(LPARAM lParam1, LPARAM lParam2,
                                     LPARAM lParamSort)
{
    int comp=0;
    DQRControl* pDQRctrl = (DQRControl*)lParamSort;
    if(!pDQRctrl)    return 0;
    // Check if we work with the parent item
    if((DWORD)lParam1 == m_ParentListItemData)    return -1;
    if((DWORD)lParam2 == m_ParentListItemData)    return 1;
    // Find item index from item data
    LVFINDINFO info;
    info.flags = LVFI_PARAM;    info.lParam = lParam1;
    int ind1 = pDQRctrl->m_ResultsList.FindItem(&info);

```

```

if(ind1 == -1) return 0;
info.flags = LVFI_PARAM; info.lParam = lParam2;
int ind2 = pDQRCtrl->m_ResultsList.FindItem(&info);
if(ind2 == -1) return 0;
// Compare the items
int col = pDQRCtrl->m_nSortColumn;
CString s1 = pDQRCtrl->m_ResultsList.GetItemText(ind1, col);
if(s1=="*" || s1=="?" || s1==" " || s1=="") return 1;
CString s2 = pDQRCtrl->m_ResultsList.GetItemText(ind2, col);
if(s2=="*" || s2=="?" || s2==" " || s2=="") return -1;
if( col == m_ColumnBDate || col == m_ColumnBTime ||
   col == m_ColumnSDate || col == m_ColumnSTime ) // dates
{
    COleDateTime dt1, dt2;
    dt1.ParseDateTime(s1);
    if(dt1.GetStatus() != COleDateTime::valid) return 1;
    dt2.ParseDateTime(s2);
    if(dt2.GetStatus() != COleDateTime::valid) return -1;
    if(dt1<dt2) comp=-1;
    else
    {
        if(dt1>dt2) comp=1;
        else comp=0;
    }
}
else if (col == m_ColumnSImgNum) // numbers
{
    int n1 = atoi(s1);
    int n2 = atoi(s2);
    if(n1<n2) comp=-1;
    else
    {
        if(n1>n2) comp=1;
        else comp=0;
    }
}
else // strings
{
    comp = s1.CompareNoCase(s2);
}
if(!pDQRCtrl->m_bSortInIncreasingOrder) comp = -comp;
return comp;
}

*****
Task-processing threads
*****/
UINT DQRControl::RunTaskQueueThread(LPVOID pCtrl)
{
    if(!pCtrl) return (UINT)FALSE;
    DQRControl* pDQRC = (DQRControl*) pCtrl;
    while(true) // task-processing loop
    {
        pDQRC->UpdateTaskView(pDQRC->m_DQR.ExecuteNextTask());
        Sleep(1000);
    }
}

bool DQRControl::StartTaskQueue()
{
    if(!m_UseTaskQueue) return true; // no queues
    return (AfxBeginThread(RunTaskQueueThread, this,
        THREAD_PRIORITY_LOWEST, m_ClientStackSize) != NULL);
}

void DQRControl::UpdateTaskView(bool reload_list)
{
    if(reload_list) m_TaskView.LoadTasksList();
    m_TaskView.UpdateClock();
}

```

```

/*****
*
*   Persistent storage
*
*****/
void DQRControl::SerializeDQRControl(bool is_loading)
{
    CString fname = theApp.app_DirectoryData;
    if(m_LocalOnly) return; //fname += CString("\\locTasks.dat");
    else                 fname += CString("\\remTasks.dat");
    FILE* fp;
    fp = fopen((char*)(LPCSTR)fname, is_loading ? "r" : "w");
    if(!fp) return;
    ::Serializebool(fp,m_PromptForTaskScheduler,is_loading);
    if(m_UseTaskQueue)
    {
        if( !is_loading )
        {
            if(m_DQR.GetTasksSize()>0 &&
                AfxMessageBox("Save your queued queries?",
                    MB_ICONQUESTION | MB_YESNO)==IDYES )
            {
                m_DQR.SerializeDQR(fp, is_loading);
            }
        }
        else m_DQR.SerializeDQR(fp, is_loading);
    }
    fclose(fp);
}

/*****
*
*   Task Scheduler
*
*****/
void DQRControl::SetTaskSchedulerPrompt(bool enable)
{
    m_PromptForTaskScheduler = enable;
    // No prompts on local or without queues
    if(m_LocalOnly || !m_UseTaskQueue) m_PromptForTaskScheduler=false;
}

bool DQRControl::GetTaskSchedulerPrompt()
{
    return m_PromptForTaskScheduler;
}

```

```

#ifndef AFX_DQRTASKVIEW_H_INCLUDED_
#define AFX_DQRTASKVIEW_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// dqrtaskview.h : header file
//

/////////////////////////////////////////////////////////////////
// DQRTaskView dialog

class DQRTaskView : public CDialog
{
// Construction
public:
    void UpdateClock();
    void LoadTasksList();
    bool AttachDQR(DQR* pDDR);
    DQRTaskView(CWnd* pParent = NULL); // standard constructor
    ~DQRTaskView() { m_ImageList.DeleteImageList(); };

// Dialog Data
   //{{AFX_DATA(DQRTaskView)
    enum { IDD = IDD_DIALOG_DQRVIEW };
    CListCtrl m_TasksList;
    }}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(DQRTaskView)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(DQRTaskView)
    virtual BOOL OnInitDialog();
    afx_msg void OnDelete();
    afx_msg void OnRescheduleTask();
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    const static int m_ColumnTasks, m_ColumnArchive, m_ColumnData,
        m_ColumnAttempts, m_ColumnSchedule;
    CImageList m_ImageList;
    DQR* m_pDQR;
};

/////////////////////////////////////////////////////////////////
// DQRTaskSchedule dialog

class DQRTaskSchedule : public CDialog
{
// Construction
public:
    void RunScheduler(DQRTask& t, bool prompt);
    void ScheduleTask(DQRTask& t);
    DQRTaskSchedule(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(DQRTaskSchedule)
    enum { IDD = IDD_DIALOG_DQRVIEW_SCHEDULE };
    CComboBox m_ComboTo;
    CComboBox m_ComboFrom;
    CTime m_EndDate;
    CTime m_EndTime;
    CTime m_StartDate;
    CTime m_StartTime;
    }}AFX_DATA

```

```
UINT    m_Attempts;
//}}AFX_DATA
```

```
// Overrides
```

```
// ClassWizard generated virtual function overrides
```

```
//{{AFX_VIRTUAL(DQRTaskSchedule)
```

```
protected:
```

```
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
```

```
//}}AFX_VIRTUAL
```

```
// Implementation
```

```
protected:
```

```
// Generated message map functions
```

```
//{{AFX_MSG(DQRTaskSchedule)
```

```
afx_msg void OnSelChangeComboFrom();
```

```
virtual BOOL OnInitDialog();
```

```
afx_msg void OnSelChangeComboTo();
```

```
virtual void OnOK();
```

```
//}}AFX_MSG
```

```
DECLARE_MESSAGE_MAP()
```

```
private:
```

```
    BYTE        m_FromIndex, m_ToIndex;
```

```
};
```

```
//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
```

```
#endif // !defined(AFX_DQRTASKVIEW_H_INCLUDED_)
```

```

// dqrtaskview.cpp : implementation file
//

#include "stdafx.h"
#include "..\DCM.h"
#include "dqrtaskview.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// DQRTaskView dialog

const int DQRTaskView::m_ColumnTasks = 0;
const int DQRTaskView::m_ColumnArchive = 1;
const int DQRTaskView::m_ColumnData = 2;
const int DQRTaskView::m_ColumnAttempts = 3;
const int DQRTaskView::m_ColumnSchedule = 4;

DQRTaskView::DQRTaskView(CWnd* pParent /*=NULL*/)
: CDialog(DQRTaskView::IDD, pParent)
{
   //{{AFX_DATA_INIT(DQRTaskView)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
    m_pDQR = NULL;
}

void DQRTaskView::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    {{{AFX_DATA_MAP(DQRTaskView)
    DDX_Control(pDX, IDC_LIST_TASKS, m_TasksList);
    }}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DQRTaskView, CDialog)
    {{{AFX_MSG_MAP(DQRTaskView)
    ON_BN_CLICKED(IDC_DELETE, OnDelete)
    ON_BN_CLICKED(IDC_RESCHEDULE, OnRescheduleTask)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// DQRTaskView message handlers
bool DQRTaskView::AttachDQR(DQR* pDQR)
{
    if(!pDQR) return false;
    m_pDQR = pDQR;
    return true;
}

BOOL DQRTaskView::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    /* SET m_TasksList : */
    // 1. Load icons
    if(m_ImageList.m_hImageList==NULL)
    {
        if(!m_ImageList.Create(16,17,ILC_COLOR4,3,1))
        { return FALSE; }
        m_ImageList.Add(theApp.LoadIcon(IDI_TASK_DOWNLOAD));
        m_ImageList.Add(theApp.LoadIcon(IDI_TASK_SCHEDULE));
    }
    m_TasksList.SetImageList(&m_ImageList,LVSIL_SMALL);

```

```

// 2. Load columns
m_TasksList.InsertColumn(m_ColumnTasks, "Task Status", LVCFMT_CENTER, 110, 0);
m_TasksList.InsertColumn(m_ColumnArchive, "Archive", LVCFMT_CENTER, 110, 0);
m_TasksList.InsertColumn(m_ColumnData, "Data", LVCFMT_CENTER, 190, 0);
m_TasksList.InsertColumn(m_ColumnAttempts, "Attempts", LVCFMT_CENTER, 60, 0);
m_TasksList.InsertColumn(m_ColumnSchedule, "Schedule", LVCFMT_CENTER, 80, 0);
m_TasksList.SetColumnWidth(m_ColumnSchedule, LVSCW_AUTOSIZE_USEHEADER);
// 3. Force entire row selection
m_TasksList.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_SUBITEMIMAGES
                             | LVS_EX_GRIDLINES);

// Display current tasks
LoadTasksList();

return TRUE; // return TRUE unless you set the focus to a control
             // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
* (Re)Load the list of tasks
*
*****/
void DQRTaskView::LoadTasksList()
{
    if(!m_pDQR || !m_TasksList.GetSafeHwnd()) return;
    m_TasksList.DeleteAllItems();
    char s[128];
    int nItem, nItemType;
    CString sItemType;
    DQRTask t;

    for(int n=m_pDQR->GetTasksSize()-1; n>=0; n--)
    {
        if(!m_pDQR->GetTask(n,t)) continue;
        // Find out the item type
        if(t.CanExecuteNow())
        {
            nItemType = 0; sItemType = " In Progress...";
        }
        else
        {
            if(t.CanExecuteLater())
            {
                nItemType = 1; sItemType = " On Schedule";
            }
            else
            {
                nItemType = 0; sItemType = " Finishing ...";
            }
        }
        nItem = m_TasksList.InsertItem(n, sItemType, nItemType);
        m_pDQR->GetAELocation(t.m_nAE, s);
        m_TasksList.SetItem(nItem, m_ColumnArchive, LVIF_TEXT, s, 0, 0, 0, 0);
        sprintf(s, "%s, %s", t.m_DRdata.GetPatientName(), t.m_DRdata.GetPatientID());
        m_TasksList.SetItem(nItem, m_ColumnData, LVIF_TEXT, s, 0, 0, 0, 0);
        sprintf(s, "%d", t.GetExec());
        m_TasksList.SetItem(nItem, m_ColumnAttempts, LVIF_TEXT, s, 0, 0, 0, 0);
        t.FormatScheduleString(s, 127);
        m_TasksList.SetItem(nItem, m_ColumnSchedule, LVIF_TEXT, s, 0, 0, 0, 0);
        m_TasksList.SetItemData(nItem, t.GetID());
    }
    // Prevent horizontal scroll
    m_TasksList.SetColumnWidth(m_ColumnSchedule, LVSCW_AUTOSIZE_USEHEADER);
    UpdateClock();
}

/*****
*
* Update clock on the dialog
*
*****/
void DQRTaskView::UpdateClock()
{

```



```

    if(!GetSafeHwnd()) return;
    // Show current date and time
    CTime tm = CTime::GetCurrentTime();
    GetDlgItem(IDC_CURRENT)
        ->SetWindowText(tm.Format("%A, %d %B %Y, %Hh %Mm %Ss"));
}
/*****
*
*   Delete selected tasks
*
*****/
void DQRTaskView::OnDelete()
{
    if(!m_pDQR) return;
    POSITION pos = m_TasksList.GetFirstSelectedItemPosition();
    if (pos == NULL) return; // nothing selected
    int sel=-1;
    UINT taskID;
    while(pos)
    {
        sel = m_TasksList.GetNextSelectedItem(pos);
        if(sel<0) continue;
        taskID = m_TasksList.GetItemData(sel);
        m_pDQR->RemoveTaskID(taskID);
    }
    LoadTasksList();
}

/*****
*
*   Reschedule selected tasks
*
*****/
void DQRTaskView::OnRescheduleTask()
{
    if(!m_pDQR) return;
    POSITION pos = m_TasksList.GetFirstSelectedItemPosition();
    if (pos == NULL) return; // nothing selected
    int sel=-1;
    UINT taskID;
    DQRTask* pTask;
    DQRTaskSchedule dts;
    if(dts.DoModal() != IDOK) return;
    while(pos)
    {
        sel = m_TasksList.GetNextSelectedItem(pos);
        if(sel<0) continue;
        taskID = m_TasksList.GetItemData(sel);
        pTask = m_pDQR->GetTaskPtrFromID(taskID);
        if(pTask) dts.ScheduleTask(*pTask);
    }
    LoadTasksList();
}

////////////////////////////////////
// DQRTaskSchedule dialog

DQRTaskSchedule::DQRTaskSchedule(CWnd* pParent /*=NULL*/)
: CDialog(DQRTaskSchedule::IDD, pParent)
{
    //{{AFX_DATA_INIT(DQRTaskSchedule)
    m_EndDate = CTime::GetCurrentTime();
    m_EndTime = CTime::GetCurrentTime();
    m_StartDate = CTime::GetCurrentTime();
    m_StartTime = CTime::GetCurrentTime();
    m_Attempts = 1;
    //}}AFX_DATA_INIT

```

```

    m_FromIndex=m_ToIndex=0;
}

void DQRTaskSchedule::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DQRTaskSchedule)
    DDX_Control(pDX, IDC_COMBO_TO, m_ComboTo);
    DDX_Control(pDX, IDC_COMBO_FROM, m_ComboFrom);
    DDX_DateTimeCtrl(pDX, IDC_END_DATE, m_EndDate);
    DDX_DateTimeCtrl(pDX, IDC_END_TIME, m_EndTime);
    DDX_DateTimeCtrl(pDX, IDC_START_DATE, m_StartDate);
    DDX_DateTimeCtrl(pDX, IDC_START_TIME, m_StartTime);
    DDX_Text(pDX, IDC_ATTEMPTS, m_Attempts);
    DDV_MinMaxUInt(pDX, m_Attempts, 1, 5);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DQRTaskSchedule, CDialog)
    //{{AFX_MSG_MAP(DQRTaskSchedule)
    ON_CBN_SELCHANGE(IDC_COMBO_FROM, OnSelChangeComboFrom)
    ON_CBN_SELCHANGE(IDC_COMBO_TO, OnSelChangeComboTo)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// DQRTaskSchedule message handlers
void DQRTaskSchedule::OnSelChangeComboFrom()
{
    m_FromIndex = m_ComboFrom.GetCurSel();
    CString txt;
    m_ComboFrom.GetWindowText(txt);
    int sw = (txt=="specific time" ? SW_NORMAL : SW_HIDE);
    GetDlgItem(IDC_START_DATE)->ShowWindow(sw);
    GetDlgItem(IDC_START_TIME)->ShowWindow(sw);
}

void DQRTaskSchedule::OnSelChangeComboTo()
{
    m_ToIndex = m_ComboTo.GetCurSel();
    CString txt;
    m_ComboTo.GetWindowText(txt);
    int sw = (txt=="specific time" ? SW_NORMAL : SW_HIDE);
    GetDlgItem(IDC_END_DATE)->ShowWindow(sw);
    GetDlgItem(IDC_END_TIME)->ShowWindow(sw);
}

BOOL DQRTaskSchedule::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_ComboFrom.SetCurSel(m_FromIndex); OnSelChangeComboFrom();
    m_ComboTo.SetCurSel(m_ToIndex); OnSelChangeComboTo();

    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void DQRTaskSchedule::OnOK()
{
    UpdateData(TRUE);
    CDialog::OnOK();
}

/*****
*
*   Write schedule information into a task
*
*****/
void DQRTaskSchedule::ScheduleTask(DQRTask &t)
{
    CTime tm;
    // 1. Find out start date/time

```

```

        // No choice was made from the listbox,
        // and the edit control was empty
        m_ListIndex=0;
        m_AEComboList.SetCurSel(0);
        UpdateAllFields(true);
    }
    else if ((n=m_AEComboList.FindStringExact(-1,info)) != CB_ERR)
    {
        // The edited string already exists
        m_ListIndex=n;
        m_AEComboList.SetCurSel(n);
        UpdateAllFields(true);
    }
    else
    {
        // Totally new string was entered
        m_AEarray->Get(m_ListIndex).SetLocation((char*)(LPCSTR)info);
        m_AEComboList.SetCurSel(m_ListIndex);
    }
}

/*****
*
*   Buttons
*
*****/
void AEOptions_Dialog::OnOK()
{
    // TODO: Add extra validation here
    UpdateAllFields(false);
    OnCloseupComboAeList();
    CDialog::OnOK();
}

void AEOptions_Dialog::OnAENew()
{
    ApplicationEntity a("<New AE Title>", 255,255,255,255);
    m_AEarray->Add(a);
    ResetAEList(m_AEarray->GetUpperBound());
}

void AEOptions_Dialog::OnAeClone()
{
    UpdateAllFields(false);
    ApplicationEntity a = m_AEarray->Get(m_ListIndex);
    CString s; s.Format("%s_Copy",a.ae_Title);
    s = s.Left(min(s.GetLength(),16));
    a.SetTitle((char*)(LPCSTR)(s));
    m_AEarray->Add(a);
    ResetAEList(m_AEarray->GetUpperBound());
}

void AEOptions_Dialog::OnAEDelete()
{
    int ind = m_AEComboList.GetCurSel();
    if(ind == (int)m_AEarray->GetLocalIndex())
    {
        AfxMessageBox("You cannot delete local AE",
            MB_ICONEXCLAMATION|MB_OK);
        return;
    }
    if(ind>=(int)(m_AEarray->GetSize()) || ind<0) return;
    m_AEarray->RemoveAt(ind);
    if(ind>0) ind--;
    ResetAEList(ind);
}

/*****
*
*   Totally reset AE list
*
*****/

```

```

void AEOptions_Dialog::ResetList(int new_selection)
{
    m_AEComboList.ResetContent();
    CString loc;
    for(UINT i=0; i<m_AEarray->GetSize(); i++)
    {
        loc=CString(m_AEarray->Get(i).ae_Location);
        if(i==m_AEarray->GetLocalIndex() && loc.Find("<Local>")<0)
        {
            loc=CString("<Local> ") + loc;
        }
        m_AEComboList.InsertString(i, loc);
    }
    if(new_selection>=(int)(m_AEarray->GetSize()) || new_selection<0)    new_selection=0;
    m_ListIndex=new_selection;
    m_AEComboList.SetCurSel(new_selection);
    UpdateAllFields(true);
}

```

```

/*****
 *
 *   Update selection index
 *
 *****/
void AEOptions_Dialog::OnSelchangeComboAeList()
{

```

```

    m_ListIndex=m_AEComboList.GetCurSel();
}

```

```

#ifndef AFX_DQRCONTROL_H_INCLUDED_
#define AFX_DQRCONTROL_H_INCLUDED_

#include "..\Controls.h" // Added by ClassView
#include "dqrtaskview.h" // Added by ClassView
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// dqrcontrol.h : header file
//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// DQRSearch dialog

class DQRSearch : public CDialog
{
// Construction
public:
    void WriteIntoDICOMRecord(DICOMRecord& dr);
    DateTimeSegment* GetBirthDatePtr();
    DQRSearch(CWnd* pParent = NULL); // standard constructor
// Dialog Data
   //{{AFX_DATA(DQRSearch)
    enum { IDD = IDD_DIALOG_DQRSEARCH };
    CString m_PatientID;
    CString m_PatientName;
    CString m_AccessionNumber;
    CString m_StudyID;
    CString m_StudyInstUID;
    CString m_Modality;
    }//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(DQRSearch)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    }//}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
   //{{AFX_MSG(DQRSearch)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    afx_msg void OnButtonAdvancedOrBasic();
    }//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    bool m_Advanced;
    DateTimeControl m_BirthDateControl, m_BirthTimeControl,
    m_StudyDateControl, m_StudyTimeControl;

    void SizeDialogArea();
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// DQRControl dialog

class DQRControl : public CDialog
{
// Construction
public:
    void SetTaskSchedulerPrompt(bool enable);
    void LoadArchiveList(UINT selection = 0);
    void DropOn(CWnd* wdrag, CWnd* wdrop);
    void UpdateAfterFind(bool find_success);
    void SetInterruptMode(bool interrupt);
    void SetPriority(BYTE priority) { m_DQR.SetPriority(priority); };
    void ShowTaskView() { if (HasTaskQueue()) m_TaskView.DoModal(); };
    bool DisplayOverControl(int controlID, CWnd *parent);

```

```

bool        GetTaskSchedulerPrompt();
bool        CreatedQDRControl(DICOMViewLog* ptrClientLog, DICOMDatabase* ptrDB,
                               bool local_only);

bool        HasTaskQueue()      { return m_UseTaskQueue      ; }
BYTE        GetPriority()        { return m_QDR.GetPriority(); };

QDRControl(CWnd* pParent = NULL);    // constructor
~QDRControl();                       // destructor

// Dialog Data
//{{AFX_DATA(QDRControl)
enum { IDD = IDD_DIALOG_QDRCONTROL };
CAnimateCtrl    m_AnimNetwork;
CButton         m_ButtonSearch;
CButton         m_ButtonCancel;
CButton         m_ButtonMoveTo;
CButton         m_ButtonGet;
CListCtrl       m_ResultsList;
CComboBox       m_MoveArchiveList;
CComboBox       m_ArchiveList;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(QDRControl)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
void OnHeaderClicked(NMHDR* pNMHDR, LRESULT* pResult);
// Generated message map functions
//{{AFX_MSG(QDRControl)
virtual BOOL OnInitDialog();
afx_msg void OnButtonStartSearch();
afx_msg void OnButtonGet();
afx_msg void OnButtonMoveTo();
afx_msg void OnDoubleClickListResults(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnSelChangeArchiveList();
afx_msg void OnSelChangeMoveArchiveList();
afx_msg void OnButtonCancel();
afx_msg void OnBeginDragListResults(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnButtonDelete();
//}}AFX_MSG
afx_msg void OnOk() {}; // ignore when Enter is pressed
afx_msg bool OnClickListResults(NMHDR* pNMHDR, LRESULT* pResult);
afx_msg void OnRightClickListResults(NMHDR* pNMHDR, LRESULT* pResult);
DECLARE_MESSAGE_MAP()

private:
bool        m_LocalOnly, m_UseTaskQueue, m_PromptForTaskScheduler;
bool        m_PreloadData;
bool        m_bSortInIncreasingOrder;
char        m_MoveDestinationAE[20];
int         m_nSortColumn;
int         m_ResultsListSelection[5];
int         m_DOBindex;
const static int m_ColumnPname, m_ColumnPid, m_ColumnAnum,
m_ColumnMod, m_ColumnSDate, m_ColumnSTime,
m_ColumnBDate, m_ColumnBTime, m_ColumnSImgNum;

const static UINT m_ClientStackSize;
const static DWORD m_ParentListItemData;
HWND        m_HWND;
CCriticalSection m_RunClientThread_CritSection;
ApplicationEntityList* m_AEarray;
CImageList   m_ImageList;
QDRSearch    m_SearchDialog;
QDR          m_QDR;
QDRTaskView  m_TaskView;
QDRTaskSchedule m_TaskScheduler;
enum
{
    echo,

```

```
find, find_root, find_previous, find_next,
get_index, move_index;
```

```
} m_RequestedClientService;
```

```
void SerializedDQRControl(bool is_loading);
void EnableCancel();
void UpdateTaskView(bool reload_list);
void DropOn(CWnd* win);
void TestArchiveConnection();
void FindAll();
static void MarkAEServerStatuses(ApplicationEntityList *ael,
                                UINT ae_index, bool status);
bool LoadFoundList(bool erase);
bool StartAllServers();
bool StartAEServer(UINT ae_index);
bool StartTaskQueue();
bool RunClientThread();
static bool GetLocalIP(CString& sname, BYTE& ip1, BYTE& ip2,
                      BYTE& ip3, BYTE& ip4);
BOOL UpdateData( BOOL bSaveAndValidate=TRUE);
int GetThreadPriority();
int GetResultsListSelection(int& sel, int& dcm_index);
static int DQRCtrlCallbackFilter(void* dqrctrl, UINT stepnum=0,
                                UINT id=0);
static int CALLBACK CompareItems(LPARAM lParam1,
                                LPARAM lParam2, LPARAM lParamSort);
static UINT StartQRServer(LPVOID index);
static UINT StartQRClient(LPVOID ptrDQRControl);
static UINT RunTaskQueueThread(LPVOID pCtrl);
CString GetLevelPrompt(bool higher);
```

```
#endif // !defined(AFX_DQRCONTROL_H_INCLUDED_)
```

```
// dqrcontrol.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\DCM.h"
#include "dqrcontrol.h"
#include "Server.h"
#include <process.h>
```

```
#ifdef _DEBUG
```

```
#define new DEBUG_NEW
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
```

```
#endif
```

```
////////////////////////////////////
```

```
// DQRSearch dialog
```

```
DQRSearch::DQRSearch(CWnd* pParent /*=NULL*/)
: CDialog(DQRSearch::IDD, pParent)
```

```
{
```

```
    //{AFX_DATA_INIT(DQRSearch)
```

```
    m_PatientID = _T("");
```

```
    m_PatientName = _T("");
```

```
    m_AccessionNumber = _T("");
```

```
    m_StudyID = _T("");
```

```
    m_StudyInstUID = _T("");
```

```
    m_Modality = _T("");
```

```
    //}AFX_DATA_INIT
```

```
    m_Advanced = false;
```

```
    m_BirthDateControl.SetDateFormat();
```

```
    m_BirthDateControl.SetTitle("Birth Date: ");
```

```
    m_BirthTimeControl.SetTimeFormat();
```

```
    m_BirthTimeControl.SetTitle("Birth Time: ");
```

```
    m_StudyDateControl.SetDateFormat();
```

```
    m_StudyDateControl.SetTitle("Study Date: ");
```

```
    m_StudyTimeControl.SetTimeFormat();
```

```
    m_StudyTimeControl.SetTitle("Study Time: ");
```

```
}
```

```
void DQRSearch::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
```

```
    //{AFX_DATA_MAP(DQRSearch)
```

```
    DDX_Text(pDX, IDC_PATIENT_ID, m_PatientID);
```

```
    DDX_Text(pDX, IDC_PATIENT_NAME, m_PatientName);
```

```
    DDX_Text(pDX, IDC_STUDY_ACCNUM, m_AccessionNumber);
```

```
    DDV_MaxChars(pDX, m_AccessionNumber, 16);
```

```
    DDX_Text(pDX, IDC_STUDY_ID, m_StudyID);
```

```
    DDV_MaxChars(pDX, m_StudyID, 16);
```

```
    DDX_Text(pDX, IDC_STUDY_INSTUID, m_StudyInstUID);
```

```
    DDV_MaxChars(pDX, m_StudyInstUID, 64);
```

```
    DDX_Text(pDX, IDC_MODALITY, m_Modality);
```

```
    DDV_MaxChars(pDX, m_Modality, 3);
```

```
    //}AFX_DATA_MAP
```

```
}
```

```
BEGIN_MESSAGE_MAP(DQRSearch, CDialog)
```

```
    //{AFX_MSG_MAP(DQRSearch)
```

```
    ON_BN_CLICKED(IDC_BUTTON_ADV, OnButtonAdvancedOrBasic)
```

```
    //}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()
```

```
////////////////////////////////////
```

```
// DQRSearch message handlers
```

```
BOOL DQRSearch::OnInitDialog()
```

```
{
```

```
    CDialog::OnInitDialog();
```

```
    // Place DateTimeSegment controls
```

```
    if(!m_BirthDateControl.DisplayOverControl(IDC_BIRTHDATE, this))
```

```
        return FALSE;
```

```
    if(!m_BirthTimeControl.DisplayOverControl(IDC_BIRTHTIME, this))
```

```
        return FALSE;
```



```

if(!m_StudyDateControl.DisplayOverControl(IDC_STDATE, this))
    return FALSE;
if(!m_StudyTimeControl.DisplayOverControl(IDC_STTIME, this))
    return FALSE;

SizeDialogArea();
return TRUE; // return TRUE unless you set the focus to a control
            // EXCEPTION: OCX Property Pages should return FALSE
}

```

```

/*****
*
*   Validating input on exit
*
*****/
void DQRSearch::OnOK()
{

```

```

    UpdateData(TRUE);
    m_BirthDateControl.UpdateData(TRUE);
    m_BirthTimeControl.UpdateData(TRUE);
    CDialog::OnOK();
}

```

```

/*****
*
*   Switch between advanced and basic parameters
*
*****/

```

```

void DQRSearch::SizeDialogArea()
{
    CWnd* pWnd = GetDlgItem(IDC_BUTTON_ADV);
    if(!pWnd) return;
    if(!m_Advanced)
    {
        CRect rc, rcb;
        pWnd->GetWindowRect(rcb);
        GetWindowRect(rc);
        //ScreenToClient(rc);
        rc.bottom = rcb.bottom+10;
        MoveWindow(rc);
        pWnd->SetWindowText("Advanced >>");
    }
    else
    {
        CWnd* pWnd2 = GetDlgItem(IDC_STTIME);
        if(!pWnd2) return;
        CRect rc, rcb;
        pWnd2->GetWindowRect(rcb);
        GetWindowRect(rc);
        //ScreenToClient(rc);
        rc.bottom = rcb.bottom+10;
        MoveWindow(rc);
        pWnd->SetWindowText("Basic <<");
    }
}

```

```

void DQRSearch::OnButtonAdvancedOrBasic()
{
    m_Advanced = !m_Advanced;
    SizeDialogArea();
}

```

```

/*****
*
*   Access DateTimeSegments
*
*****/

```

```

DateTimeSegment* DQRSearch::GetBirthDatePtr()
{
    return &(m_BirthDateControl.GetDateTimeSegment());
}

```

```

/*****
*
*   Fill DICOMRecord with the data from this dialog

```

```

*
*****/
void DQRSearch::WriteIntoDICOMRecord(DICOMRecord &dr)
{
    if(!m_Advanced) // baseline search
    {
        dr.SetRecord( (char*)(LPCSTR)m_PatientID,
            (char*)(LPCSTR)m_PatientName,
            &(m_BirthDateControl.GetDateTimeSegment()),
            &(m_BirthTimeControl.GetDateTimeSegment()),
            NULL,
            NULL, NULL, NULL,
            NULL, NULL,
            NULL, NULL, NULL,
            NULL, NULL, NULL);
    }
    else
    {
        dr.SetRecord( (char*)(LPCSTR)m_PatientID,
            (char*)(LPCSTR)m_PatientName,
            &(m_BirthDateControl.GetDateTimeSegment()),
            &(m_BirthTimeControl.GetDateTimeSegment()),
            (char*)(LPCSTR)m_StudyInstUID,
            (char*)(LPCSTR)m_StudyID,
            (char*)(LPCSTR)m_AccessionNumber, NULL,
            &(m_StudyDateControl.GetDateTimeSegment()),
            &(m_StudyTimeControl.GetDateTimeSegment()),
            NULL, (char*)(LPCSTR)m_Modality, NULL,
            NULL, NULL, NULL);
    }
}

```

```

// DQRControl.cpp : implementation file

```

```

const int DQRControl::m_ColumnPname = 0;
const int DQRControl::m_ColumnPid = 1;
const int DQRControl::m_ColumnBDate = 2;
const int DQRControl::m_ColumnSDate = 3;
const int DQRControl::m_ColumnSTime = 4;
const int DQRControl::m_ColumnMod = 5;
const int DQRControl::m_ColumnSImgNum = 6;
const int DQRControl::m_ColumnAnum = 7;
const int DQRControl::m_ColumnBTime = 8;
const UINT DQRControl::m_ClientStackSize=1000;
const DWORD DQRControl::m_ParentListItemData=9999999;

```

```

////////////////////////////////////
// DQRControl dialog

```

```

DQRControl::DQRControl(CWnd* pParent /*=NULL*/)
: CDialog(DQRControl::IDD, pParent)
{
    //{{AFX_DATA_INIT(DQRControl)
    //}}AFX_DATA_INIT
    m_HWND = NULL;
    m_AEarray = NULL;
    m_LocalOnly=false; m_UseTaskQueue=true; // Queue tasks on remote
    m_PromptForTaskScheduler = true; // Prompt for task scheduler on remote
    m_PreloadData = false;
    m_bSortInIncreasingOrder = true;
    m_nSortColumn = 0;
}
DQRControl::~DQRControl()

```

```

{
    m_ImageList.DeleteImageList();
    if(!m_LocalOnly && m_AEarray)
    {
        m_AEarray->SetCurrentIndex(m_DQR.GetCurrentAEIndex());
    }
    SerializedDQRControl(false);
}

void DQRControl::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DQRControl)
    DDX_Control(pDX, IDC_ANIMATE_NETWORKING, m_AnimNetwork);
    DDX_Control(pDX, IDC_BUTTON_START_SEARCH, m_ButtonSearch);
    DDX_Control(pDX, IDC_BUTTON_CANCEL, m_ButtonCancel);
    DDX_Control(pDX, IDC_BUTTON_MOVETO, m_ButtonMoveTo);
    DDX_Control(pDX, IDC_BUTTON_GET, m_ButtonGet);
    DDX_Control(pDX, IDC_LIST_RESULTS, m_ResultsList);
    DDX_Control(pDX, IDC_COMBO_MOVEARCHIVE, m_MoveArchiveList);
    DDX_Control(pDX, IDC_COMBO_ARCHIVE, m_ArchiveList);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DQRControl, CDialog)
    //{{AFX_MSG_MAP(DQRControl)
    ON_BN_CLICKED(IDC_BUTTON_START_SEARCH, OnButtonStartSearch)
    ON_BN_CLICKED(IDC_BUTTON_GET, OnButtonGet)
    ON_BN_CLICKED(IDC_BUTTON_MOVETO, OnButtonMoveTo)
    ON_NOTIFY(NM_DBLCLK, IDC_LIST_RESULTS, OnDoubleClickListResults)
    ON_CBN_SELCHANGE(IDC_COMBO_ARCHIVE, OnSelChangeArchiveList)
    ON_CBN_SELCHANGE(IDC_COMBO_MOVEARCHIVE, OnSelChangeMoveArchiveList)
    ON_BN_CLICKED(IDC_BUTTON_CANCEL, OnButtonCancel)
    ON_NOTIFY(LVN_BEGINDRAG, IDC_LIST_RESULTS, OnBeginDragListResults)
    ON_NOTIFY(NM_CLICK, IDC_LIST_RESULTS, OnClickListResults)
    ON_NOTIFY(NM_RCLICK, IDC_LIST_RESULTS, OnRightClickListResults)
    ON_BN_CLICKED(IDC_BUTTON_DELETE, OnButtonDelete)
    //}}AFX_MSG_MAP
    ON_COMMAND(IDOK, OnOk)
    ON_COMMAND(IDCANCEL, OnOk)
    ON_NOTIFY(HDN_ITEMCLICKA, 0, OnHeaderClicked) // column sort
    ON_NOTIFY(HDN_ITEMCLICKW, 0, OnHeaderClicked) // column sort
END_MESSAGE_MAP()

//DQRControl message handlers

/*****
*
*   Display the control
*
*****/
bool DQRControl::DisplayOverControl(int controlID, CWnd *parent)
{
    if(!parent || !controlID) return false;
    CWnd* pWnd = parent->GetDlgItem(controlID);
    if (pWnd)
    {
        CRect rc;
        pWnd->GetWindowRect(rc);
        parent->ScreenToClient(rc);
        this->Create(IDD, parent);
        this->SetWindowPos(pWnd, rc.left, rc.top, 0, 0, SWP_NOSIZE|SWP_SHOWWINDOW);
        return true;
    }
    return false;
}

/*****
*
*   Get string for the current QR level
*
*****/

```

```

*****/
CString DQRControl::GetLevelPrompt(bool higher)
{
    CString prompt;
    BYTE lev = m_DQR.GetLevel();
    if(!higher)
    {
        if(lev == DICOMRecord::LevelPatient)    prompt = "PATIENT";
        else if(lev == DICOMRecord::LevelStudy) prompt = "STUDY";
        else if(lev == DICOMRecord::LevelSeries) prompt = "SERIES";
        else if(lev == DICOMRecord::LevelImage) prompt = "IMAGE";
        else prompt = "";
    }
    else
    {
        if(m_DQR.IsOnRootLevel())    prompt=" Start new search";
        else if(lev == DICOMRecord::LevelStudy)    prompt = " Go to patient level";
        else if(lev == DICOMRecord::LevelSeries)    prompt = " Go to study level";
        else if(lev == DICOMRecord::LevelImage)    prompt = " Go to series level";
        else prompt = "";
    }
    return prompt;
}

/*****
*
*   (Re)Load the list of archives
*
*****/
void DQRControl::LoadArchiveList(UINT selection /*=0*/)
{
    if(!m_AEarray)    return;
    CString loc;
    m_ArchiveList.ResetContent();    m_MoveArchiveList.ResetContent();
    for(UINT i=0; i<m_AEarray->GetSize(); i++)
    {
        loc=CString(m_AEarray->Get(i).ae_Location);
        if(i==m_AEarray->GetLocalIndex() && loc.Find("<Local>")<0)
        {
            loc = CString("<Local> ") + loc;
        }
        m_ArchiveList.InsertString(i,loc);
        m_MoveArchiveList.InsertString(i,loc);
    }

    if(m_LocalOnly)    selection = m_AEarray->GetLocalIndex();
    m_DQR.SetCurrentAEIndex(selection);
    selection=m_DQR.GetCurrentAEIndex();
    m_ArchiveList.SetCurSel(selection);
    if(selection>0)    selection=0;    else    selection=1;
    m_MoveArchiveList.SetCurSel(selection);
    if(m_LocalOnly)
    {
        m_ArchiveList.GetWindowText(loc);
        loc += CString(" archive");
        GetDlgItem(IDC_ARCHIVE_LOCATION)->SetWindowText(loc);
        m_ArchiveList.ShowWindow(SW_HIDE);
    }
    UpdateData(FALSE);
}

/*****
*
*   On new selection in the archive list
*
*****/
void DQRControl::OnSelChangeArchiveList()
{
    if(!m_AEarray && m_LocalOnly)    return;
    UINT n=m_ArchiveList.GetCurSel();
    if(n!=m_DQR.GetCurrentAEIndex())    LoadFoundList(true);
    if(!m_DQR.SetCurrentAEIndex(n))
    {
        UINT m = m_DQR.GetCurrentAEIndex();
        if(m>n)

```

```

    {
        m=0;
        m_DQR.SetCurrentAEIndex(0);
    }
    m_ArchiveList.SetCurSel(m);
    LoadFoundList(true);
}
TestArchiveConnection();
// Make sure current archive is different from "Move to" archive
n=m_ArchiveList.GetCurSel();
if(m_MoveArchiveList.GetCurSel()==(int)n)
{
    if(n>0) n=0; else n=1;
    m_MoveArchiveList.SetCurSel(n);
}
}
void DQRControl::OnSelChangeMoveArchiveList()
{
    if(!m_AEarray) return;
    int n = m_ArchiveList.GetCurSel();
    if(n>=(int)m_AEarray->GetSize())
    {
        LoadArchiveList(); return;
    }
    if(m_MoveArchiveList.GetCurSel() == n)
    {
        CString s;
        m_ArchiveList.GetWindowText(s);
        CString info;
        info.Format("You are currently connected to %s\n"
            " and cannot use it as move destination.", s);
        AfxMessageBox(info, MB_ICONINFORMATION);
        if(n>0) n=n-1; else n=1;
        m_MoveArchiveList.SetCurSel(n);
    }
}

/*****
*
* (Re)Load the list of found DICOM data objects
*
*****/
bool DQRControl::LoadFoundList(bool erase)
{
    if(!GetSafeHwnd()) return false;
    CString label;
    CString lev=GetLevelPrompt(false);
    CString res_lev=CString(" on ")+lev+CString(" level");
    lev.MakeLower();
    m_ButtonGet.EnableWindow(FALSE);
    label = m_LocalOnly ? "Open " : "Download ";
    m_ButtonGet.SetWindowText(label+lev);
    m_ButtonMoveTo.EnableWindow(FALSE);
    label = m_LocalOnly ? "Upload " : "Move ";
    m_ButtonMoveTo.SetWindowText(label+lev+ CString(" to: "));
    m_ResultsList.DeleteAllItems();
    CString res("Results: no matches");
    if(erase)
    {
        m_DQR.ClearFound();
        m_ResultsListSelection[0]=1;
        m_ResultsListSelection[1]=1;
        m_ResultsListSelection[2]=1;
        m_ResultsListSelection[3]=1;
        GetDlgItem(IDC_RESULTS_FRAME)->SetWindowText(res+res_lev);
        UpdateData(FALSE);
        return true;
    }
    int nFound = m_DQR.GetFoundCount();
    m_ResultsList.SetItemCount(nFound+1);
    GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText(
        nFound>0 ? "Loading data..." : "");
    char s[64];
    int nItem, nI;

```

```

CString str;
DICOMRecord dr;
for(nI=nFound-1; nI>=0; nI--)
{
    m_DQR.GetFoundRecord(dr,nI);
    // Patient name
    str=CString(dr.GetPatientName()); str.Replace('^',' ');
    nItem=m_ResultsList.InsertItem(nI,str,max(1,m_DQR.GetLevel())-1);
    // Patient ID
    m_ResultsList.SetItem(nItem,m_ColumnPid,LVIF_TEXT,
        dr.GetPatientID(),0,0,0,0);
    // Accession number
    m_ResultsList.SetItem(nItem,m_ColumnAnum,LVIF_TEXT,
        dr.GetAccessionNumber(),0,0,0,0);
    // Modality
    m_ResultsList.SetItem(nItem,m_ColumnMod,LVIF_TEXT/*|LVIF_IMAGE*/,
        dr.GetModality(),0,0,0,0);
    // Study Date
    dr.FormatStudyDate(s,64,false);
    m_ResultsList.SetItem(nItem,m_ColumnSDate,LVIF_TEXT,s,0,0,0,0);
    // Study Time
    dr.FormatStudyTime(s,64,false);
    m_ResultsList.SetItem(nItem,m_ColumnSTime,LVIF_TEXT,s,0,0,0,0);
    // Birth Date
    dr.FormatPatientBirthDate(s,64,false);
    m_ResultsList.SetItem(nItem,m_ColumnBDate,LVIF_TEXT,s,0,0,0,0);
    // Number of study related images
    m_ResultsList.SetItem(nItem,m_ColumnSImgNum,LVIF_TEXT,
        dr.GetStudyImagesNum(),0,0,0,0);
    // Associate array index with item data, to allow for list sorting
    m_ResultsList.SetItemData(nItem,nI);
    // Birth Time - do not use
    //dr.FormatPatientBirthTime(s,64,false);
    //m_ResultsList.SetItem(nItem,m_ColumnBTime,LVIF_TEXT,s,0,0,0,0);
}
// "Higher level" item
nItem=m_ResultsList.InsertItem(0,(const char*)(GetLevelPrompt(true)),4);
m_ResultsList.SetItemData(nItem,m_ParentListItemData);
// Prevent horizontal scroll
m_ResultsList.SetColumnWidth(m_ColumnAnum,LVSCW_AUTOSIZE_USEHEADER);
// Results label
if(nFound>0)
{
    if(nFound==1) res.Format("Results: 1 match found");
    else res.Format("Results: %d matches found",nFound);
    res += res_lev;
}
GetDlgItem(IDC_RESULTS_FRAME)->SetWindowText(res);
GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText("");
// Size columns to their data
/*
for(int i=0; i<=6; i++)
{
    if(i!=m_ColumnPname && i!=m_ColumnPid) continue;
    m_ResultsList.SetColumnWidth(i,LVSCW_AUTOSIZE_USEHEADER);
}
*/
UpdateData(FALSE);
return true;
}

/*****
*
* Get current selection from the Results list
* and corresponding found object index
* Returns number of parameters successfully retrieved
*
*****/
int DQRControl::GetResultsListSelection(int& sel, int& dcm_index)
{
    sel=dcm_index=-1;
    POSITION pos = m_ResultsList.GetFirstSelectedItemPosition();
    int result;
    if (pos == NULL) result=0; // nothing selected

```

```

else
{
    sel = m_ResultsList.GetNextSelectedItem(pos); //single selection
    dcm_index=(int)(m_ResultsList.GetItemData(sel));
    if(dcm_index<0 || dcm_index>=m_DQR.GetFoundCount())
    {
        dcm_index=-1;    result=1;
    }
    else
    {
        result=2;
    }
}
m_ButtonGet.EnableWindow(result>1); m_ButtonMoveTo.EnableWindow(result>1);
return result;
}
/*****
*
*   Clicks on Results list
*
*****/
bool DQRControl::OnClickListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    if(pResult) *pResult = 0;
    int sel, dcm;
    bool item=(GetResultsListSelection(sel, dcm)==2);
    return item;
}

void DQRControl::OnDoubleClickListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    *pResult = 0;
    int sel, dcm;
    int item=GetResultsListSelection(sel, dcm);
    int nlev=max(1,m_DQR.GetLevel()-1);
    switch (item)
    {
        case 1:// go to higher level
            if(m_DQR.IsOnRootLevel())    OnButtonStartSearch();
            else
            {
                m_RequestedClientService=find_previous;
                if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
                    m_ClientStackSize)) RunClientThread();
            }
            return;
        case 2:// go to lower level
            if(m_DQR.IsOnBottomLevel()) { OnButtonGet(); return; }
            else
            {
                m_ResultsListSelection[nlev]=sel;
                m_DOBindex = dcm;
                m_RequestedClientService=find_next;
                if(!AfxBeginThread(StartQRClient,this,GetThreadPriority(),
                    m_ClientStackSize)) RunClientThread();
                return;
            }
    }
}

return;
}

void DQRControl::OnRightClickListResults(NMHDR* pNMHDR, LRESULT* pResult)
{
    if(!OnClickListResults(pNMHDR, pResult)) return;
    // Click positioning
    CPoint p(GetMessagePos());
    m_ResultsList.ScreenToClient(&p);

    // Create popup menu
    CMenu menu;
    menu.CreatePopupMenu();
    CMenu menu1;
    menu1.LoadMenu(IDR_MENU_DQRCONTROL);
    CMenu* pop=menu1.GetSubMenu(1);

    // Set acceptable menu IDs

```

```

UINT ids[2]={ IDC_BUTTON_GET,
              IDC_BUTTON_MOVE_TO};

// Create the popup menu
int n=0;
unsigned int nid;
CString strMenu, remAET;
for(unsigned int i=0; i<pop->GetMenuItemCount(); i++)
{
    nid=pop->GetMenuItemID(i);
    if( nid!=ids[0] && nid!=ids[1] ) continue;
    if(nid==IDC_BUTTON_GET) m_ButtonGet.GetWindowText(strMenu);
    else if(nid==IDC_BUTTON_MOVE_TO)
    {
        m_ButtonMoveTo.GetWindowText(strMenu); strMenu.TrimRight(" :");
        m_MoveArchiveList.GetWindowText(remAET);
        strMenu += CString(" ") + remAET;
    }
    else pop->GetMenuString(i, strMenu, MF_BYPOSITION);
    int status=pop->GetMenuState(i, MF_BYPOSITION);
    menu.InsertMenu(n, status, nid, strMenu);
    n++;
}

// Display popup menu
ClientToScreen(&p);
p=CPoint(p.x,p.y);
menu.TrackPopupMenu(TPM_LEFTALIGN, p.x,p.y,this);
pop->DestroyMenu();
menu.DestroyMenu();
menu1.DestroyMenu();

*****
* Start the dialog
*****/
BOOL DQRControl::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_HWND=GetSafeHwnd();
    LoadArchiveList(m_DQR.GetCurrentAEIndex());
    LoadFoundList(true);
    m_ButtonCancel.ShowWindow(SW_HIDE);
    m_AnimNetwork.ShowWindow(SW_HIDE);
    GetDlgItem(IDC_STATIC_PROGRESS)->ShowWindow(SW_HIDE);
    if(!m_LocalOnly)
    {
        GetDlgItem(IDC_BUTTON_DELETE)->ShowWindow(SW_HIDE);
        GetDlgItem(IDC_ARCHIVE_LOCATION)->SetWindowPos(this, 0, 0, 90, 14,
            SWP_NOMOVE | SWP_NOZORDER | SWP_SHOWWINDOW);
    }
    else
        GetDlgItem(IDC_CONNECTION)->ShowWindow(SW_HIDE);

    /* SET m_ResultsList : */
    // 1. Load icons
    if(m_ImageList.m_hImageList==NULL)
    {
        if(!m_ImageList.Create(16,17,ILC_COLOR4,5,1))
        {
            return FALSE;
        }
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_PATIENT));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_STUDY));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_SERIES));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_IMAGE));
        m_ImageList.Add(theApp.LoadIcon(IDI_ICON_LEVEL_HIGHER));
    }
    m_ResultsList.SetImageList(&m_ImageList, LVSIL_SMALL);
    // 2. Load columns
    m_ResultsList.InsertColumn(m_ColumnPname, "Patient Name", LVCFMT_CENTER, 120, 0);
    m_ResultsList.InsertColumn(m_ColumnPid, "Patient ID", LVCFMT_CENTER, 80, 0);
    m_ResultsList.InsertColumn(m_ColumnBDate, "Birth Date", LVCFMT_CENTER, 80, 0);

```



```

m_ResultsList.InsertColumn(m_ColumnSDate,"Study Date", LVCFMT_CENTER,80,0);
m_ResultsList.InsertColumn(m_ColumnSTime,"Study Time", LVCFMT_CENTER,80,0);
m_ResultsList.InsertColumn(m_ColumnMod,"Modality", LVCFMT_CENTER,60,0);
m_ResultsList.InsertColumn(m_ColumnSImgNum,"Images", LVCFMT_CENTER,50,0);
m_ResultsList.InsertColumn(m_ColumnAnum,"Access. #", LVCFMT_LEFT,100,0);
//m_ResultsList.InsertColumn(m_ColumnBTime,"Birth Time", LVCFMT_CENTER,80,0);
m_ResultsList.SetColumnWidth(m_ColumnAnum,LVSCW_AUTOSIZE_USEHEADER);
// 3. Force entire row selection
m_ResultsList.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_SUBITEMIMAGES
    | LVS_EX_GRIDLINES);
// Load all data if required
FindAll();
return TRUE; // return TRUE unless you set the focus to a control
            // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
*   Set log files (client and server)
*
*****/
bool DQRControl::CreatedQRControl(DICOMViewLog* ptrClientLog, DICOMDatabase* ptrDB,
    bool local_only)
{
    static bool getIP=true, startServers=true;
    if(!ptrClientLog || !ptrDB)
    {
        AfxMessageBox("Cannot initialize Query/Retrieve control with NULL parameters");
        return false;
    }
    // Set parameters
    if(!m_DQR.CreateQR(ptrClientLog, ptrDB))
    {
        AfxMessageBox("Failed to initialize Query/Retrieve control");
        return false;
    }
    m_DQR.SetDQRCallBack(DQRCtrlCallbackFilter, this);
    m_TaskView.AttachDQR(&m_DQR);
    m_LocalOnly=local_only;
    m_UseTaskQueue = !m_LocalOnly; // Queue tasks on remote only
    m_PromptForTaskScheduler = m_UseTaskQueue; // Prompt for schedule by default
    m_PreloadData = m_LocalOnly; // Preload patient data on local
    m_AEarray = m_DQR.GetAEListPtr();
    // Find local AE at first call
    if(getIP)
    {
        BYTE ip1, ip2, ip3, ip4;
        CString comp_name;
        if(GetLocalIP(comp_name, ip1,ip2,ip3,ip4))
        {
            if(!m_AEarray->SetLocalAE(ip1,ip2,ip3,ip4, (char*)(LPCSTR)comp_name))
            {
                AfxMessageBox("Cannot find Application Entity for this PC");
                return false;
            }
        }
        getIP=false;
    }
    if(!StartTaskQueue())
    {
        AfxMessageBox("Failed to initialize background task queue");
        return false;
    }
    // Start all servers at first call
    if(startServers)
    {
        StartAllServers();
        startServers=false;
    }
    // Retrieve serialized task data
    SerializeDQRControl(true);
    return true;
}

```

```

/*****
*
*   Test network connection to current remote archive (C-ECHO)
*
*****/
void DQRControl::TestArchiveConnection()
{
    Beep(500,100);
    m_RequestedClientService=echo;
    if( !AfxBeginThread(StartQRClient,this,
        GetThreadPriority(),m_ClientStackSize)) RunClientThread();
}

/*****
*
*   Delete. Works with local database only !
*
*****/
void DQRControl::OnButtonDelete()
{
    if(!m_LocalOnly)    return;
    int sel=-1, dcm=-1;
    DICOMRecord dr;
    // Anything selected ?
    if(GetResultsListSelection(sel, dcm) == 2)
    {
        if(!m_DQR.GetFoundRecord(dr,dcm))    return;
    }
    else
    {
        DQRSearch ds;
        if(ds.DoModal() != IDOK)    return;
        ds.WriteIntoDICOMRecord(dr);
    }
    if(AfxMessageBox("Delete specified items from the local database ?",
        MB_YESNO) != IDYES) return;
    int n = m_DQR.DeleteFromLocalDB(dr);
    if(n>0) // Repeat last Find to refresh the worklist window
    {
        m_RequestedClientService=find;
        if( !AfxBeginThread(StartQRClient,this,
            GetThreadPriority(),m_ClientStackSize)) RunClientThread();
    }
}

/*****
*
*   Cancel last network CFind, CGet or CMove
*
*****/
void DQRControl::OnButtonCancel()
{
    Beep(500,100);
    m_DQR.Cancel();
    GetDlgItem(IDC_STATIC_PROGRESS)->SetWindowText("Cancelling requested, wait ...");
}

void DQRControl::EnableCancel()
{
    if(m_RequestedClientService != echo)
    {
        m_ButtonCancel.ShowWindow(SW_SHOW);
    }
}

/*****
*
*   Start search
*
*****/
void DQRControl::OnButtonStartSearch()
{

```

```

long Image::TR_Sobel(const int x, const int y)
{
    long a00=GetLuminance(x-1,y-1); long a10=GetLuminance(x,y-1);
    long a20=GetLuminance(x+1,y-1); long a01=GetLuminance(x-1,y);
    /*long a11=GetLuminance(x,y); */
    long a21=GetLuminance(x+1,y); long a12=GetLuminance(x,y+1);
    long a02=GetLuminance(x-1,y+1); long a22=GetLuminance(x+1,y+1);

    long tx=(a22-a02)+(a20-a00)+2*(a21-a01);
    long ty=(a00-a02)+(a20-a22)+2*(a10-a12);
    long t=(long) (sqrt(tx*tx+ty*ty));
    if(t<m_EdgeThreshold) t=0;
    if(t>m_maxPixelValue) t=m_maxPixelValue;
    return t;
}

/*****
*
*   Apply Hurst fractal operator
*
*****/
long Image::TR_Fractal(const int x, const int y)
{
    long p, pmin, pmax;
    double t, s1=0.0, s2=0.0;

    // Find log variance for each distance from the central pixel
    // d=1
    p=GetLuminance(x-1,y); pmax=pmin=p;
    p=GetLuminance(x+1,y); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += 0.0; s2 += t;
    // d=sqrt(2)
    p=GetLuminance(x-1,y-1); pmax=pmin=p;
    p=GetLuminance(x+1,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-1,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+1,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*0.34657359; s2 += t; //we use ln(sqrt(2));
    // d=2
    p=GetLuminance(x-2,y); pmax=pmin=p;
    p=GetLuminance(x+2,y); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y-2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*0.69314718; s2 += t;
    // d=sqrt(5)
    p=GetLuminance(x-1,y-2); pmax=pmin=p;
    p=GetLuminance(x-1,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+1,y-2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+1,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-2,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-2,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+2,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+2,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*0.80471896; s2 += t;
    // d=sqrt(8)
    p=GetLuminance(x-2,y-2); pmax=pmin=p;
    p=GetLuminance(x+2,y-2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x-2,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x+2,y+2); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*1.03972077; s2 += t;
    // d=3
    p=GetLuminance(x-3,y); pmax=pmin=p;
    p=GetLuminance(x+3,y); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y-3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    p=GetLuminance(x,y+3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
    t=log(max(pmax-pmin,1));
    s1 += t*1.09861229; s2 += t;
}

```

```

// d=sqrt(10)
p=GetLuminance(x-1,y-3); pmax=pmin=p;
p=GetLuminance(x-1,y+3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+1,y-3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+1,y+3); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x-3,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x-3,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+3,y-1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
p=GetLuminance(x+3,y+1); if(p>pmax) pmax=p; else if(p<pmin) pmin=p;
t=log(max(pmax-pmin,1));
s1 += t*1.15129255; s2 += t;
s1 *= 0.14285714; s2 = s2*0.10477684;

//Find fractal dimension as slope of the Hurst line
//t=max(6.386491206*(s1-s2),0.0); // fractal dimension
t=max(1000*(s1-s2),0.0); // fractal dimension
return( (long)(t) );
}

/*****
*
* Apply a pixel neighborhood function (with code mask_type) to the image
*
*****/
bool Image::TR_PixelNeighborhood(char mask_type, bool show_progress)
{
    int rad,i,j,j1,top,bot;
    long p, pmin, pmax;
    double tr_scale=1.0;

    // Set pointer to the masking function
    long (Image::*maskF)(const int x, const int y) = 0;
    switch(mask_type)
    {
        case 'g': maskF=TR_DeNoise;rad=2; break; // denoising
        case 'a': maskF=TR_Smooth; rad=1; break; // average smoothing
        case 's': maskF=TR_Sharp; rad=1; break; // sharpening
        case 'e': maskF=TR_Sobel; rad=1; break; // edge detector
        case 'f': maskF=TR_Fractal; rad=3; break; // fractal dimension
        default: return false; // invalid mask type
    }

    // Create temporary buffer
    int w = GetWidth();
    int h = GetHeight();
    long *(*buf)=new long*[rad+1];
    if(!buf)
    {
        AfxMessageBox("Low memory, cannot transform");
        return false;
    }
    for(i=0; i<=rad; i++)
    {
        buf[i] = new long[w];
        if(buf[i]==0)
        {
            AfxMessageBox("Low memory, cannot transform");
            for(j=0; j<i; j++) { if(buf[j]) delete [] buf[j]; }
            delete [] buf;
            return false;
        } // out of memory
    }

    // Display progress control in the main frame status bar
    if(show_progress) theApp.ShowProgress(5,"Transforming ...");

    // Find transform scaling factor
    if(mask_type!='e' && mask_type!='f')
    {
        pmin=0; tr_scale=1.0;
    }
    else
    {

```

```

// Estimate transform pixel value range
p=pmin=pmax=(this->*maskF)(rad+1,rad+1);
int dw = w>>3;  if(dw<1) dw=1;
int dh = h>>3;  if(dh<1) dh=1;
for(i=rad+1; i<w-rad; i += dw)
{
    for(j=rad+1; j<h-rad; j += dh)
    {
        p=(this->*maskF)(i,j);
        if(p>pmax) pmax=p;
        else if(p<pmin) pmin=p;
    }
    pmax++;
    tr_scale = (double)(m_maxPixelValue)/(pmax-pmin);
}
if(show_progress) theApp.ShowProgress(10);

// Backup pixel values
ResetPixels(true);

// Apply (2*rad-1)*(2*rad-1) masking operator
bot=-1;
top=rad-1;
int procent=0;
for(j=rad; j<=h; j++)
{
    if(show_progress && j%50 == 0)
    {
        procent = 10+(90*j)/h;
        if(procent%3==0) theApp.ShowProgress(procent);
    }
    j1=j-rad-1;
    if(j1>=rad)
    {
        for(i=rad; i<w-rad; i++) SetPixel(i,j1,buf[bot][i]);
    }
    bot = (bot+1)%(rad+1);
    top = (top+1)%(rad+1);
    if(j<h-rad)
    {
        for(i=rad; i<w-rad; i++)
        {
            buf[top][i]=(long)( tr_scale*( (this->*maskF)(i,j)-pmin ) );
        }
    }
}

// Clean up
for(j=0; j<=rad; j++) { if(buf[j]) delete [] buf[j]; }
delete [] buf;
theApp.ShowProgress(0);
Beep(500,50);
return true;
}

```

```

/*****
*
*   Inverts the image
*
*****/
bool Image::TR_Negate()
{
    if(m_pPal->p_active) m_pPal->Negate();
    else

```

```

{
    // Display progress control in the main frame status bar
    theApp.ShowProgress(1,"Changing to negative image ...");
    // Backup pixel values
    ResetPixels(true);
    // Invert pixels values
    if(m_RGB)
    {
        BYTE r, g, b;
        long p;
        for(long i=0; i<(long)m_numPixels; i++)
        {
            if(i%1000==0) theApp.ShowProgress((99*i)/m_numPixels);
            p=GetPixel(i);
            r=m_maxPixelValue-GetRValue(p);
            g=m_maxPixelValue-GetGValue(p);
            b=m_maxPixelValue-GetBValue(p);
            SetPixel(i,RGB(r,g,b));
        }
    }
    else
    {
        for(long i=0; i<(long)m_numPixels; i++)
        {
            if(i%1000==0) theApp.ShowProgress((99*i)/m_numPixels);
            SetPixel(i,m_maxPixelValue-GetPixel(i));
        }
    }
}
theApp.ShowProgress(0);
Beep(500,100);
return true;

```

```

*****
Copy pixels from current to safe buffer (on true)
or vice versa (on false)

```

```

*****/
void Image::ResetPixels(bool current_to_safe)
{
    if(current_to_safe) // current -> safe
    {
        if(m_UndoFile != "+") return;
        m_UndoFileCount++;
        m_UndoFile.Format("%s\\_pix%04d.tmp",theApp.app_DirectoryTmp,m_UndoFileCount);
        FILE* fp = fopen(m_UndoFile,"wb");
        if(!fp) { m_UndoFile = "+"; return; }
        //fwrite(m_Pixels,1,m_numPixelBytes,fp);
        SerializeImage(fp, false);
        fclose(fp);
    }
    else // safe -> current
    {
        if(m_UndoFile == "+" || m_UndoFile == "") return;
        FILE* fp = fopen(m_UndoFile,"rb");
        if(!fp) { return; }
        //fread(m_Pixels,1,m_numPixelBytes,fp);
        SerializeImage(fp, true);
        fclose(fp);
        m_UpdateBitmap=true;
    }
}

```

```

/*****
*
* Complete image serialization into a binary file
*
*****/

```

```

bool Image::SerializeImage(FILE *fp, bool is_loading)
{
    int width, height, bpp;
    if(!is_loading)
    {
        width = m_Width;
        height = m_Height;
        bpp = m_Bytes_per_Pixel;
    }

    if(!::SerializeInteger(fp, width, is_loading)) return false;
    if(!::SerializeInteger(fp, height, is_loading)) return false;
    if(!::SerializeInteger(fp, bpp, is_loading)) return false;
    if(is_loading)
    {
        // reformat the image if needed
        if(width!=m_Width || height!=m_Height || bpp!=m_Bytes_per_Pixel)
        {
            if(!CreateImage(width, height, bpp)) return false;
        }
    }
    if(!m_ScreenMap.SerializeScreenMap(fp, is_loading)) return false;
    if(!m_DICOMRecord.SerializeDICOMRecord(fp, is_loading)) return false;
    if(!is_loading) fwrite(m_Pixels,1,m_numPixelBytes,fp);
    else fread(m_Pixels,1,m_numPixelBytes,fp);
    return true;
}

*****
Histogramm stretch from [amin,amax] to [bmin,bmax] color range.
R,G and B components are stretched together
*****/
bool Image::TR_HistStretch(int amin,int amax,int bmin,int bmax,
                           bool show_progress)
{
    long i, cmax, p;

    /* Create color map */
    if(m_pPal->p_active) cmax=m_pPal->p_Size;
    else Get_Pixel_minmax(i,cmax);
    cmax++;
    long* color_map;
    try { color_map=new long[cmax]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot perform this transform",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot perform this transform",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Set color map parameters */
    if(amin<0) amin=0;
    if(bmin<0) bmin=0;
    if(amax>m_maxPixelValue) amax=m_maxPixelValue;
    if(bmax>m_maxPixelValue) bmax=m_maxPixelValue;
    if (amin>=amax || bmin>=bmax) // invalid map
    {
        delete [] color_map; return false;
    }
    if(amin==bmin && amax==bmax) // no stretch needed
    {
        delete [] color_map; return true;
    }
}

```

```

/* Fill the color map */
long da=amax-amin;
long db=bmax-bmin;
for(i=0; i<cmax; i++)
{
    p=bmin+(db*(i-amin))/da;
    if(p<bmin) p=bmin; else if (p>bmax) p=bmax;
    color_map[i]=p;
}

```

```
SetPalette(color_map, cmax, show_progress);
```

```
delete [] color_map;
return true;
}

```

```

/*****
 *
 * Histogramm stretch from (percent)% median neighborhood
 * to the maximal [0,m_maxPixelValue] range
 *
 *****/
bool Image::TR_HistStretch(BYTE percent, bool show_progress)
{
    long i, amin, amax, amed, p;

    /* Validation */
    if (percent>=100) percent=99;
    if(percent<0) return true;

    /* Find pixel statistics */
    Get_Pixel_minmax(amin,amax);
    if(percent==0) return TR_HistStretch(amin,amax,0,
                                         m_maxPixelValue,show_progress); // simple stretch

    /* Initialize image histogram */
    long cmax=amax+1;
    int* hist;
    try { hist=new int[cmax]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot perform this transform",
                      MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!hist)
    {
        AfxMessageBox("Low memory, cannot perform this transform");
        return false;
    } // out of memory
    for(i=0; i<cmax; i++) hist[i]=0;

    /* Estimate image histogram */
    int hmax=20000;
    int di=__max(1,m_numPixels/10000);
    if(m_RGB) // Color image
    {
        for(i=0; i<m_numPixelBytes; i += di)
        {
            p=m_Pixels[i];
            if(hist[p]<hmax) hist[p] += di;
        }
    }
    else // Greyscale image
    {
        for(i=0; i<(long)m_numPixels; i += di)
        {
            p=GetPixel(i);
            if(hist[p]<hmax) hist[p] += di;
        }
    }
}

```



```

/* Find histogram color range */
double ptot=0, tot=0;
for(i=0; i<cmax; i++)
{
    ptot += ((double)i)*hist[i];
    tot += hist[i];
}
amed = (long)(0.5+ptot/tot);

/* Find new intensity range to preserve */
double keep_max=(100-percent)*tot/100; // number of pixels to keep
double keep=hist[amed];
long bmin=amed; long bmax=amed;
do // do at least once to guarantee bmin<bmax
{
    if(bmin>amin)
    {
        bmin--;
        keep += hist[bmin];
    }
    if(bmax<amax)
    {
        bmax++;
        keep += hist[bmax];
    }
} while (keep<=keep_max);
delete [] hist;

if( ((bmin!=amin)|| (bmax!=amax)) && (bmin<bmax) )
    return TR_HistStretch(bmin,bmax,0,m_maxPixelValue,show_progress);
else return false;

```

# \*\*\*\*\* Histogramm equalization \*\*\*\*\*

```

bool Image::TR_HistEqualize(bool show_progress)

```

```

    long i, amin, amax, p;

```

```

/* Find pixel statistics */
Get_Pixel_minmax(amin,amax);
Beep(300,100);

```

```

/* Initialize image histogram, also used as color map */

```

```

long cmax=amax+1;
long* hist=new long[cmax];
if(!hist)

```

```

{
    AfxMessageBox("Low memory, cannot perform this transform");
    return false;
} // out of memory
for(i=0; i<cmax; i++) hist[i]=0;

```

```

/* Compute image histogram */

```

```

long hmax=2000000;
int di=__max(1,m_numPixels/10000);
if(m_RGB) // Color image

```

```

{
    for(i=0; i<m_numPixelBytes; i += di)
    {
        p=m_Pixels[i];
        if(hist[p]<hmax) hist[p] += di;
    }
}

```

```

else // Greyscale image

```

```

{
    for(i=0; i<(long)m_numPixels; i += di)
    {
        p=GetPixel(i);

```

```

        if(hist[p]<hmax) hist[p] += di;
    }

    /* Integrate the histogram */
    double httotal=0;
    for(i=0; i<cmax; i++) httotal += hist[i];
    if(httotal<1) // did we have negative pixels or empty image ?
    {
        delete [] hist; return false;
    }
    if(hist[0]<httotal-1) { httotal -= hist[0]; hist[0]=0; }

    /* Fill the color map */
    double hcum=0;
    for(i=0; i<cmax; i++)
    {
        hcum += hist[i]; // update cumulative hist
        hist[i]=(long)((m_maxPixelValue*hcum)/httotal);
    }

    /* Remap the pixel data */
    SetPalette(hist, cmax, show_progress);

    /* Clean up */
    delete [] hist;
    return true;
}

*****
Image Gamma correction
*****/
bool Image::TR_GammaCorrection(double gamma)

    long i, p, pmax;

    /* Validation */
    if(gamma<=0.1) gamma=0.1;
    if(fabs(gamma-1.0)<0.1) return true; // gamma is nearly 1.0, no correction

    /* Find maximum pixel value */
    Get_Pixel_minmax(i,pmax);
    if(pmax==0) return true; //blank image, no correction

    /* Create color map */
    long* color_map;
    try { color_map=new long[pmax+1]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot run gamma correction",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot run gamma correction",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Fill the color map */
    double c=1.0/pmax;
    double gamma_1=1.0/gamma;
    for(i=1; i<=pmax; i++)
    {
        p=(int)(0.5+pmax*pow(c*i,gamma_1));
        if(p>pmax) color_map[i]=pmax;
        else color_map[i]=p;
    }

```

```

    }
    color_map[0]=0;

    // Display progress control in the main frame status bar
    theApp.ShowProgress(1,"Performing gamma correction ...");

    /* Remap the pixel data */
    SetPalette(color_map, pmax+1, true);

    /* Clean up */
    delete [] color_map;
    theApp.ShowProgress(0);
    Beep(500,100);
    return true;
}

/*****
 *
 *   Log transform to enhance dark images
 *
 *****/
bool Image::TR_PixelLog()
{
    long i, p, pmax;

    /* Find maximum pixel value */
    Get_Pixel_minmax(i,pmax);
    if(pmax<=2) return true; //blank image, no correction

    /* Create color map */
    long* color_map;
    try { color_map=new long[pmax+1]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot enhance dark image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot enhance dark image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Fill the color map */
    double c=m_maxPixelValue/log((double)pmax);
    for(i=0; i<=pmax; i++)
    {
        p=(long)( 0.5+c*log((double)(i+1)) );
        if(p>m_maxPixelValue) color_map[i]=m_maxPixelValue;
        else color_map[i]=p;
    }

    /* Remap the pixel data */
    SetPalette(color_map, pmax+1, true);

    /* Clean up */
    delete [] color_map;
    Beep(500,100);
    return true;
}

/*****
 *
 *   Exp transform to enhance light images
 *
 *****/

```

```

*****
bool Image::TR_PixelExp()
{
    long i, p, pmax;

    /* Find maximum pixel value */
    Get_Pixel_minmax(i, pmax);
    if(pmax<=2) return true; //blank image, no correction

    /* Create color map */
    long* color_map;
    try { color_map=new long[pmax+1]; }
    catch(...)
    {
        AfxMessageBox("Low memory, cannot enhance bright image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!color_map)
    {
        AfxMessageBox("Low memory, cannot enhance bright image",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    } // out of memory

    /* Fill the color map */
    double c=1.0/m_maxPixelValue;
    for(i=0; i<=pmax; i++)
    {
        p=(long)( pow((double)m_maxPixelValue,c*i) );
        if(p>m_maxPixelValue) color_map[i]=m_maxPixelValue;
        else color_map[i]=p;
    }

    /* Remap the pixel data */
    SetPalette(color_map, pmax+1, true);

    /* Clean up */
    delete [] color_map;
    Beep(500,100);
    return true;
}

*****
* Rotate the image
*
*****
bool Image::TR_Rotate(int degrees)
{
    degrees %= 360;
    if(degrees == 0) return true;
    if(degrees != 90 && degrees != 180 && degrees != 270) return false;
    int x, y, w, h;
    // Do we have 180 degrees ?
    if(degrees == 180)
    {
        // Backup
        ResetPixels(true);
        // Central symmetry
        w = GetWidth(); h = GetHeight();
        long p;
        for(x=0; x<w/2; x++)
        {
            ::ShowProgress((200*x)/w, "Rotating by 180 degrees...");
            for(y=0; y<h; y++)
            {
                p = GetPixel(x,y);
                SetPixel(x,y,GetPixel(w-x,h-y));
                SetPixel(w-x,h-y,p);
            }
        }
    }
}

```

```

    }
    if(w%2==0)
    {
        x = (w/2);
        for(y=0; y<h/2; y++)
        {
            p = GetPixel(x,y);
'G42Xn          um<m_NumberOfFrames; imagenum++)

{
    info.Format("Loading image # %d/%d",imagenum+1,m_NumberOfFrames);
    // Allocate next image frame
    pImg = &(Add());    if(!pImg)    break;
    if(!(pImg->CreateImage(m_Width,m_Height,(m_BitsAllocated+7)/8,
                           m_Palette)))
    {
        AfxMessageBox("Out of memory for image data", MB_ICONEXCLAMATION|MB_OK);
        RemoveLast();
        pImg = NULL;
        break;
    }
    // Load pixels
    int    percent=0;
    long    pmax=Get(0).m_maxPixelValue; // set acceptable pixel max.
    double    aprog=imagenum*m_Height;
    for(UINT y=0; y<m_Height; y++)
    {
        if(y%60==0)
        {
            percent = (int)((y+aprog)*kprog);
            if(percent%2==0) ::ShowProgress(percent,(char*)(LPCSTR)info);
        }
        for(UINT x=0; x<m_Width; x++)
        {
            p=rpd.GetBufferedPixel(i);    i++;
            pImg->SetPixelFromLum(x,y,(pmax*(p-p_min))/dp );
        }
    }
    // Set record data
    pImg->SetImageData( &dr, &m_PixelSpacing, &imagenum);
    // Create series palette
    if(imagenum==0) // first image
    {
        m_Palette = new Palette(theApp.app_Metheus,
                                pImg->m_maxPixelValue,
                                pImg->m_RGB);
        pImg->LinkToPalette(m_Palette);
    }
}
vr->Reset();    // Clear data from the vr

// Update image series parameters
if(GetSize()>0)
{
    m_Width=Get(0).GetWidth();
    m_Height=Get(0).GetHeight();
    m_SamplesPerPixel=1;    // we merged multiple samples
    m_BitsAllocated=8*Get(0).GetBytesPerPixel();
    m_BitsStored=m_HighBit=m_BitsAllocated;
}
SetShowSeriesImageInfo(true);
theApp.ShowProgress(0);

return true;
}

```

```

/*****
*
*   Save bitmap image array as VR data
*   This function is required and overrides abstract virtual
*   in the base class

```

```

*
*****
bool ImageSeries::WritePixels(VR *vr)
{
    int i;
    UINT32 data_size, dsize;
    // Find total data size
    int imagenum=GetUpperBound()+1;
    if(imagenum<=0) return true;
    data_size=0;
    for(i=0; i<imagenum; i++)
    {
        Get(i).GetPixelBytes(dsize);
        data_size += dsize;
    }

    // Allocate pixel encoder
    RawPixelEncoder rpe;
    if(!rpe.SetSize(data_size))
    {
        AfxMessageBox("Out of memory on saving pixel data",
            MB_ICONEXCLAMATION|MB_OK);
        return false;
    }

    // Copy image data into the encoder buffer
    for(i=0; i<imagenum; i++)
    {
        BYTE* data=Get(i).GetPixelBytes(dsize);
        rpe.AddData(data, dsize, Get(i).GetBytesInRow());
    }

    // Attach encoder buffer to the vr
    rpe.TransferDataToVR(vr);

    return true;
}

*****
*
* Get image pointer (safe), and update m_CurrentImageIndex
*
*****
Image* ImageSeries::GetImage(int n)
{
    // Validate image index
    if(!HasData())
    {
        m_CurrentImageIndex=-1;
        return NULL; // no images
    }
    if(n<0) n=0;
    else if (n>GetUpperBound()) n=GetUpperBound();
    // Retrieve image pointer
    m_CurrentImageIndex=n;
    return &(Get(m_CurrentImageIndex));
}

Image* ImageSeries::GetCurrentImage()
{
    return GetImage(m_CurrentImageIndex);
}

Image* ImageSeries::GetImageFromScreenPoint(CPoint &p)
{
    Image* img = GetCurrentImage();
    if(!img) return NULL;
    if(m_DisplayColumns == 0) return img;
    if(img->ContainsScreenPoint(p)) return img;
    for(int n=0; n<(int)GetSize(); n++)
    {
        if(n==m_CurrentImageIndex) continue;
        if(Get(n).ContainsScreenPoint(p))
        {
            img = GetImage(n);
            SetSelectedImage();
        }
    }
}

```

```

        break;
    }
    return img;
}

/*****
 *
 *   Sets selection to current image
 *
 *****/
void ImageSeries::SetSelectedImage()
{
    for(unsigned int n=0; n<GetSize(); n++)
    {
        if(Get(n).GetDisplayStatus() != Image::DisplaySelected) continue;
        Get(n).SetDisplayStatus(Image::DisplayNormal);
    }
    Image* img = GetCurrentImage();
    if(img) img->SetDisplayStatus(Image::DisplaySelected);
}

/*****
 *
 *   Adding new DICOMObjects to the series
 *
 *****/
bool ImageSeries::AddDDO(DICOMDataObject &ddo, bool destroy_original)
{
    bool success = true;
    bool read = !HasData();
    ImageSeries tmp_series;
    if(destroy_original) // we can destroy ddo
    {
        if(read) success=ReadDO(ddo);
        else success=tmp_series.ReadDO(ddo);
    }
    else // we must keep the original => use clone
    {
        DICOMDataObject ddo_tmp;
        success = ddo_tmp.CloneFrom(&ddo);
        if(success)
        {
            if(read) success=ReadDO(ddo_tmp);
            else success=tmp_series.ReadDO(ddo_tmp);
        }
    }
    if(!success)
    {
        AfxMessageBox("Cannot add DICOM object to the series",
            MB_ICONEXCLAMATION|MB_OK);
        return false;
    }
    if(read) return success; // no need to append
    else return AddSeries(tmp_series,true);
}

bool ImageSeries::AddDDOFile(CString filename)
{
    DICOMDataObject ddo;
    // Load DDO completely, but without RTC
    if( ddo.LoadFromFile((char*)(LPCSTR)filename, false) )
    {
        return AddDDO(ddo, true);
    }
    else return false;
}

void ImageSeries::AddDICOMRecords(Array<DICOMRecord>& a)
{
    int n=0;
    while (n<(int)a.GetSize())
    {
        if(BelongsToThisSeries(a[n]))
        {

```

```

        AddDDOFile(a[n].FileName());
        a.RemoveAt(n);
    }
    else n++;
}
}
/*****
*
*   Adding new series to "this" series, assuming "this" is not empty
*
*****/
bool ImageSeries::AddSeries(ImageSeries &s, bool delete_s)
{
    int ns=s.GetUpperBound();
    if(ns<0)    return true;// nothing to do
    int n = GetUpperBound();
    if(n<0) return true;    // do not add to empty !

    if(delete_s)    Array<Image>::Include(s);
    else            SetSize(n+1+ns+1);
    for(int i=0; i<=ns; i++)
    {
        int k=n+i+1;
        if(!delete_s)    Get(k).CloneFrom(&s[i]);
        if(k>0)
        {
            Get(k).LinkToPalette(Get(0).m_pPal);
        }
    }

    // Force layout update
    SetDisplayColumns(DisplayColumnsAutomatic);
    SetShowSeriesImageInfo(GetShowSeriesImageInfo());

    return true;
}

/*****
*
*   Test if dr represents an object that belongs to the same series
*   as (this)
*
*****/
bool ImageSeries::BelongsToThisSeries(DICOMRecord &dr)
{
    if(GetSize()<=0)    return true;
    return (Get(0).CompareToDICOMRecord(dr, DICOMRecord::LevelSeries) == 0);
}

/*****
*
*   Save the series as a set of BMP files
*   in the given directory
*
*****/
bool ImageSeries::SaveAsImageFiles(CString &directory, CString prefix)
{
    if(!HasData())    return true;
    // Set validated directory and prefix
    directory.TrimLeft();    directory.TrimRight();
    if(!::CreateAndSetCurrentDirectory((char*)(LPCSTR)directory))
    {
        AfxMessageBox("Cannot set image directory\n"+directory,
            MB_ICONEXCLAMATION|MB_OK);
        return false;
    }
    prefix.Replace("\\\\/, .?!*_ ", NULL);
    if(prefix=="")    prefix="img";
    prefix += "_";
    prefix=directory+CString("\\")+prefix;

    // Store image series
    CString num;
    for(int i=0; i<=GetUpperBound(); i++)

```



```

    {
        num.Format("%s%03d", prefix, i+1);
        if(!Get(i).WriteToFile(num))
        {
            return false;
        }
    }
    return true;
}

/*****
 *
 *   Set image info for all images in the series
 *
 *****/
void ImageSeries::SetShowSeriesImageInfo(bool show)
{
    for(unsigned int i=0; i<GetSize(); i++) Get(i).SetShowImageInfo(show);
}

bool ImageSeries::GetShowSeriesImageInfo()
{
    if(GetSize()<1) return false;
    return Get(0).GetShowImageInfo();
}

/*****
 *
 *   Display images
 *
 *****/
Image* ImageSeries::DisplayImages(CDC *pDC, CPoint pScroll/*=CPoint(0,0)*/)
{
    // Grab current image pointer
    Image* pImg = GetCurrentImage();
    if(!pImg) return NULL; // no images in this series

    // Initialize screen rectangle, if needed
    if(m_ScreenRect.Width() == 0) SetScreenRect(1.0);

    // Compute image layout, if needed
    if(m_DisplayColumnsChanged)
    {
        SetOptimalImageLayout();
        m_DisplayColumnsChanged=false;
    }

    // Display
    if(m_DisplayColumns == 0) // Single image
    {
        pImg->DisplayDIB(pDC, pScroll);
    }
    else
    {
        for(unsigned int n=0; n<GetSize(); n++)
        {
            Get(n).DisplayDIB(pDC, pScroll);
        }
    }
    return pImg;
}

/*****
 *
 *   Set the number of display columns.
 *
 *****/
bool ImageSeries::SetDisplayColumns(int ncol)
{
    m_DisplayColumnsChanged = (ncol != m_DisplayColumns);
    m_DisplayColumns = ncol;
    return m_DisplayColumnsChanged;
}

/*****

```

```

*
* Find the optimal multi-image layout.
*
*****/
void ImageSeries::SetOptimalImageLayout()
{
    int icount=0, n;
    // Process individual image display
    if(m_DisplayColumns == 0)
    {
        for(unsigned n=0; n<GetSize(); n++)
        {
            Get(n).m_ScreenMap.FitInsideScreenRect(m_ScreenRect);
        }
        return;
    }
    // Process multiple image display
    for(n=0; n<(int)GetSize(); n++)
    {
        if(Get(n).GetDisplayStatus() == Image::DisplayHidden) continue;
        icount ++;
    }
    if(icount<1) return; // nothing to display
    int nr, best_nc;
    Image* pImg = GetCurrentImage();
    if(!pImg) return;
    int iw = pImg->GetWidth();
    int ih = pImg->GetHeight();
    const static int d = 16;

    // Find the best image zooming factor
    double best_zoom;
    if(m_DisplayColumns>=0) // Column number was specified
    {
        best_nc = max(1,min(m_DisplayColumns,icount));
        nr = (icount+best_nc-1)/best_nc;
        best_zoom = min( (m_ScreenRect.Width()-d*(best_nc+1.0))/(iw*best_nc),
                        (m_ScreenRect.Height()-d*(nr+1.0))/(ih*nr) );
    }
    else // Column number unknown, find best
    {
        double zoom;
        best_zoom = 0.0;
        for(int nc=1; nc <= 1+icount/2; nc++)
        {
            nr = (icount+nc-1)/nc;
            zoom = min( (m_ScreenRect.Width()-d*(nc+1.0))/(iw*nc),
                      (m_ScreenRect.Height()-d*(nr+1.0))/(ih*nr) );
            if(zoom>best_zoom)
            {
                best_zoom = zoom; best_nc = nc;
            }
        }
        m_DisplayColumns = best_nc;
    }

    // Find image screen sizes
    nr = (icount+best_nc-1)/best_nc;
    int w = max(4, (int)(iw*best_zoom) );
    int h = max(4, (int)(ih*best_zoom) );
    int dx = max(1, (m_ScreenRect.Width()-w*best_nc)/(best_nc+1) );
    int dy = max(1, (m_ScreenRect.Height()-h*nr)/(nr+1) );

    // Set screen viewing rectangles for images
    int x, y;
    CRect r;
    for(n=0; n<(int)GetSize(); n++)
    {
        if(Get(n).GetDisplayStatus() == Image::DisplayHidden) continue;
        x = dx+(n % best_nc)*(w+dx);
        y = dy+(n / best_nc)*(h+dy);
        r.SetRect(x,y,x+w,y+h);
        Get(n).m_ScreenMap.FitInsideScreenRect(r);
    }
}

```

```

    return;
}

/*****
 *
 *   Alternate between browse and non-browse states
 *
 *****/
void ImageSeries::SwitchBrowseView()
{
    if(m_DisplayColumns == 0)    SetDisplayColumns(DisplayColumnsAutomatic);
    else                        SetDisplayColumns(0);
}

/*****
 *
 *   Set screen rectangle from a zoom factor
 *
 *****/
void ImageSeries::SetScreenRect(double zoom/*=1.0*/)
{
    if(zoom<0.1 || zoom > 10.0) return;
    // Find screen size
    ((CMDIFrameWnd*)AfxGetMainWnd()->MDIGetActive()->GetClientRect(m_ScreenRect);
    m_ScreenRect.BottomRight().y -= 20;
    m_ScreenRect.BottomRight().x -= 20;
    if(zoom!=1.0)    // zoom
    {
        int w = max(64, (int)(zoom*m_ScreenRect.Width()));
        int h = max(64, (int)(zoom*m_ScreenRect.Height()));
        int x = max(0, (m_ScreenRect.Width()-w)/2);
        int y = max(0, (m_ScreenRect.Height()-h)/2);
        m_ScreenRect.SetRect(x,y,x+w,y+h);
    }
    SetOptimalImageLayout();
}

/*****
 *
 *   Zoom and offset images
 *
 *****/
void ImageSeries::ZoomImages(double zoom)
{
    for(unsigned int n=0; n<GetSize(); n++)
    {
        Get(n).SetImageRectZoom(zoom);
    }
}

void ImageSeries::OffsetImages(double dx, double dy)
{
    for(unsigned int n=0; n<GetSize(); n++)
    {
        Get(n).SetImageRectOffset(dx, dy);
    }
}

double ImageSeries::GetZoom()
{
    Image* pImg = GetCurrentImage();
    if(pImg)    return pImg->GetZoom();
    else        return 1.0; // no zoom
}

```

```
// Palette.h: interface for the Palette class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_PALETTE_H_INCLUDED_
#define AFX_PALETTE_H_INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class Palette
{
public:
    bool        p_active;
    int         p_Size;

    void        BackUp();
    void        Negate(bool RGB=false);
    void        Get_Pal_minmax(long& pmin, long& pmax, bool RGB=false);
    bool        CreateNewPalette(bool Metheus, long max_color, bool rgb);
    bool        CloneFromPalette(Palette *pPal);
    bool        IsCorrectPalette();
    bool        LoadPalette(CDC* pDC, bool RGB=false);
    bool        SetPalette();
    bool        SetPalette(long* color_map, long color_map_size);
    bool        SetPalette(long offset, double stretch);
    inline long GetPaletteColor(long index);

    Palette();
    Palette(bool Metheus, long max_color, bool rgb);
    ~Palette();

private:
    bool        p_Metheus;
    bool        p_update;
    int         p_factor;
    long        p_maxCol;
    USHORT*     p_Val;
    USHORT*     p_Val_backup;
    HPALETTE    p_Hpal;

    void        DeletePalette();
};

/*****
 * Return palette color at the given index
 *****/
inline long Palette::GetPaletteColor(long index)
{
    if(index<0)            index=0;
    else if(index>=p_Size) index=p_Size-1;
    return p_Val[index];
}

#endif // !defined(AFX_PALETTE_H_INCLUDED_)
```

```
// Palette.cpp: implementation of the Palette class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "..\DCM.h"
#include "Palette.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////
Palette::Palette()
{
    p_Val = NULL; p_Val_backup = NULL; p_Hpal = NULL;
    p_update=true;    p_Metheus = false;
    p_factor = 1;     p_maxCol = 255;
    p_active = false; p_Size = 0;
}

Palette::Palette(bool Metheus, long max_color, bool rgb)
{
    p_Val = NULL; p_Val_backup = NULL; p_Hpal = NULL;
    CreateNewPalette(Metheus, max_color, rgb);
}

Palette::~Palette()
{
    DeletePalette();
}

void Palette::DeletePalette()
{
    if(p_Val) delete [] p_Val;
    if(p_Val_backup) delete [] p_Val_backup;
    if(p_Hpal) ::DeleteObject(p_Hpal);

    p_Val = NULL; p_Val_backup = NULL; p_Hpal = NULL;
    p_update=true; p_Metheus = false;
    p_factor = 1; p_maxCol = 255;
    p_active = false; p_Size = 0;
}

/* *****
 *
 * Reset palette
 *
 * *****/
bool Palette::CreateNewPalette(bool Metheus, long max_color, bool rgb)
{
    DeletePalette();
    p_Metheus=Metheus; p_factor=1; p_maxCol=max_color;
    if(p_Metheus)
    {
        p_Size=max_color+1; p_factor=USHRT_MAX/(p_Size-1);
    }
    else
    {
        if(!theApp.app_SupportedPaletteSize)
        {
            p_Size=128; p_factor=2; // No support for 8-bit palettes
        }
        else
        {
            p_Size=256; p_factor=1;
        }
    }
    if (p_maxCol>=p_Size) p_maxCol=p_Size-1;
    p_Val=0; p_Val_backup=0;
    try
    {

```

```

    p_Val_backup=new USHORT[p_Size];
    if(p_Val_backup) p_Val_backup=new USHORT[p_Size];
    p_active = (p_Val!=NULL);
}
catch(...)
{
    AfxMessageBox("Low memory, cannot allocate image palette",
        MB_OK|MB_ICONEXCLAMATION);
    p_active=false;
    return false;
}
if(p_active)
{
    for(USHORT i=0; i<p_Size; i++) p_Val[i]=i;
    BackUp();
}
p_update=true;

// Only greyscale palettes
if(p_active) p_active=!rgb;
return true;
}

/*****
*
*   Copying palettes
*
*****/
bool Palette::CloneFromPalette(Palette *pPal)
{
    if(!pPal) return false;
    if( !CreateNewPalette(pPal->p_Metheus, pPal->p_maxCol,
        !(pPal->p_active)) ) return false;
    if(pPal->p_Val)
    {
        memcpy(p_Val, pPal->p_Val, p_Size*(sizeof USHORT));
        BackUp();
    }
    return true;
}

/*****
*
*   Set identity palette
*
*****/
bool Palette::SetPalette()
{
    if(!p_active || !p_Val) return false;
    for(USHORT i=0; i<p_Size; i++) p_Val[i]=i;
    p_update=true;
    return true;
}

/*****
*
*   General (from array) palette update
*
*****/
bool Palette::SetPalette(long *color_map, long color_map_size)
{
    if(!p_active || !p_Val) return false;

    long i;
    long imax= ( p_Size>color_map_size ? color_map_size : p_Size ) ;
    for(i=0; i<imax; i++) p_Val[i]=(USHORT)labs(color_map[i]);
    for(i=imax; i<p_Size; i++) p_Val[i]=p_Val[imax-1];
    p_update=true;
    return true;
}

/*****
*
*   Linear palette update for Fast Color/Contrast
*****/

```

```

*
*****/
bool Palette::SetPalette(long offset, double stretch)
{
    if(!p_active || !p_Val) return false;

    long i, n;
    long l_offset=offset<<10;
    long l_stretch = (long)(1024*stretch);
    for(i=0; i<p_Size; i++)
    {
        n=(l_offset+((long)p_Val_backup[i])*l_stretch);
        if(n<=0) p_Val[i]=0;
        else
        {
            n >>= 10;
            if(n>=p_maxCol) n=p_maxCol-1;
            p_Val[i]=(USHORT)(n);
        }
    }
    p_update=true;
    return true;
}

/*****
*
* Create and load palette into the given DC
*
*****/
bool Palette::LoadPalette(CDC *pDC, bool RGB)
{
    if(!p_Val) return false; // cannot allocate palette
    if(!p_active) return true; // disabled palettes
    long i;
    if(!p_Metheus) // load Windows palette
    {
        if(!p_update) // same palette as before, skip palette reload
        {
            ::SelectPalette(pDC->m_hDC, p_Hpal, TRUE);
            pDC->RealizePalette();
            p_update=false;
            return true;
        }
        BYTE r,g,b;
        int cPalette = sizeof(LOGPALETTE)+sizeof(PALETTEENTRY)*p_Size;
        LOGPALETTE* pPal = (LOGPALETTE*)new BYTE[cPalette];
        if(!pPal) return false;
        pPal->palVersion = 0x300;
        pPal->palNumEntries = (unsigned short)p_Size;
        for(i = 0; i<p_Size; i++)
        {
            if(RGB)
            {
                r=GetRValue(p_Val[i]);
                g=GetGValue(p_Val[i]);
                b=GetBValue(p_Val[i]);
            }
            else r=g=b=p_factor*(BYTE)__min(p_Val[i],p_Size-1);
            pPal->palPalEntry[i].peRed = r;
            pPal->palPalEntry[i].peGreen = g;
            pPal->palPalEntry[i].peBlue = b;
            pPal->palPalEntry[i].peFlags = NULL;
        }
        DeleteObject(p_Hpal); // free display memory
        p_Hpal = CreatePalette(pPal);
        delete [] (BYTE*)pPal;
        if(!p_Hpal) return false;
        ::SelectPalette(pDC->m_hDC, p_Hpal, FALSE);
        pDC->RealizePalette();
        DeleteObject(p_Hpal); // free display memory
    }
    else // Metheus device

```

```
{
    if(!p_update) return true; //same palette as before
    USHORT* p_Val_tmp = new USHORT[p_Size];
    if(!p_Val_tmp) return false;
    for(i=0; i<p_Size; i++) p_Val_tmp[i]=p_factor*p_Val[i];
    if(MetheusLoadGrayPalette(pDC->GetSafeHdc(),
        theApp.app_DynamicPaletteStart,p_Size,p_Val_tmp)==FALSE)
    {
        delete [] p_Val_tmp;
        return false;
    }
    delete [] p_Val_tmp;
}
p_update=false;
return true;
}
```

\*\*\*\*\*

\* Find min and max palette colors

\*\*\*\*\*

void Palette::Get\_Pal\_minmax(long &pmin, long &pmax, bool RGB)

```
{
    long i, p;
    pmin=0; pmax=1;
    if(!p_Val) return;
    if(!RGB) // greyscale
    {
        pmin=p_Val[0]; pmax=pmin+1;
        for(i=1; i<p_Size; i++)
        {
            if(p_Val[i]>pmax) pmax=p_Val[i];
            else if(p_Val[i]<pmin) pmin=p_Val[i];
        }
    }
    else
    {
        BYTE r,g,b;
        pmin=GetRValue(p_Val[0]); pmax=pmin+1;
        for(i=0; i<p_Size; i++)
        {
            p=p_Val[i];
            r=GetRValue(p); g=GetGValue(p); b=GetBValue(p);
            if(r>pmax) pmax=r;
            else if(r<pmin) pmin=r;
            if(g>pmax) pmax=g;
            else if(g<pmin) pmin=g;
            if(b>pmax) pmax=b;
            else if(b<pmin) pmin=b;
        }
    }
}
```

\*\*\*\*\*

\* Invert palette colors

\*\*\*\*\*

void Palette::Negate(bool RGB)

```
{
    long i, p;
    if(!p_active || !p_Val) return;
    if(!RGB)
    {
        for(i=0; i<p_Size; i++)
        {
            p=p_maxCol-1-(long)p_Val[i];
            if(p<0) p=0; else if(p>=p_maxCol) p=p_maxCol-1;
            p_Val[i]=(USHORT)p;
        }
    }
}
```



```

    }
    else
    {
        BYTE r,g,b;
        for(i=0; i<p_Size; i++)
        {
            r=GetRValue(p_Val[i]); g=GetGValue(p_Val[i]); b=GetBValue(p_Val[i]);
            p_Val[i]=(USHORT)RGB(255-r,255-g,255-b);
        }
    }
    p_update=true;
}

/*****
*
*   Return "true" if palette supports the same color range as the image
*
*****/
bool Palette::IsCorrectPalette()
{
    return (p_factor==1 || p_Metheus);
}

/*****
*
*   Save current palette colors
*
*****/
void Palette::BackUp()
{
    if(!p_active || !p_Val || !p_Val_backup || !p_Size) return;
    memcpy(p_Val_backup, p_Val, p_Size*(sizeof USHORT));
}

```

```

// ScreenMap.h: interface for the ScreenMap class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_SCREENMAP_H_C4AA3744_77E9_11D2_9586_00105A21774F__INCLUDED_
#define AFX_SCREENMAP_H_C4AA3744_77E9_11D2_9586_00105A21774F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class ScreenMap
{
public:
    CRect    crScreen;
    CRect    crImage;

    void      FitInsideScreenRect(CRect screen);
    void      ValidateZoom(double& x);
    void      Rotate(int degrees);
    void      SetLeftTopScreenPoint(int x, int y);
    bool      Initialize(const CRect cr);
    bool      Screen_in_Image(const CPoint screenP, const int bound=0);
    bool      Screen_in_Image(const CRect screenR, const int bound=0);
    bool      SerializeScreenMap(FILE* fp, bool is_loading);
    double    GetZoom();
    CSize     GetScreenCenteredSize();
    CPoint    Image_to_Screen(const CPoint cpI);
    CPoint    Image_to_Screen(const CPoint cpS, const CPoint offset);
    CPoint    Screen_to_Image (const CPoint cpS);
    CPoint    Screen_to_Image (const CPoint cpS, const CPoint offset);
    CRect     Image_to_Screen (const CRect crI);
    CRect     Image_to_Screen (const CRect crI, const CPoint offset);
    CRect     Screen_to_Image(const CRect crS);
    CRect     Screen_to_Image(const CRect crS, const CPoint offset);

    ScreenMap();
    ScreenMap(ScreenMap& sm);
    ~ScreenMap();

private:
#endif // !defined(AFX_SCREENMAP_H_C4AA3744_77E9_11D2_9586_00105A21774F__INCLUDED_)

```

```

// ScreenMap.cpp: implementation of the ScreenMap class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "..\DCM.h"
#include "ScreenMap.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////

ScreenMap::ScreenMap()
{
}

ScreenMap::ScreenMap(ScreenMap& sm)
{
    crImage = sm.crImage;
    crScreen = sm.crScreen;
}

ScreenMap::~ScreenMap()
{
}

bool ScreenMap::Initialize(const CRect cr)
{
    crImage=cr;
    crScreen=cr;
    return true;
}

CPoint ScreenMap::Screen_to_Image(const CPoint cpS)
{
    CPoint p;
    p = cpS-crScreen.TopLeft();
    p.x= (p.x*crImage.Width())/crScreen.Width();
    p.y= (p.y*crImage.Height())/crScreen.Height();
    return (p+crImage.TopLeft());
}

CPoint ScreenMap::Screen_to_Image(const CPoint cpS, const CPoint offset)
{
    return Screen_to_Image(cpS-offset);
}

CRect ScreenMap::Screen_to_Image(const CRect crS)
{
    return CRect(this->Screen_to_Image(crS.TopLeft()),
        this->Screen_to_Image(crS.BottomRight()) );
}

CRect ScreenMap::Screen_to_Image(const CRect crS, const CPoint offset)
{
    CRect r=crS;
    r.OffsetRect(offset);
    return Screen_to_Image(r);
}

CPoint ScreenMap::Image_to_Screen(const CPoint cpI)
{
    CPoint p;
    p = cpI-crImage.TopLeft();
    p.x= (p.x*crScreen.Width())/crImage.Width();
    p.y= (p.y*crScreen.Height())/crImage.Height();

```

```

    return (p+crScreen.TopLeft());
}
CPoint ScreenMap::Image_to_Screen(const CPoint cpS, const CPoint offset)
{
    return Image_to_Screen(cpS)+offset;
}

CRect ScreenMap::Image_to_Screen(const CRect crI)
{
    return CRect(this->Image_to_Screen(crI.TopLeft()),
        this->Image_to_Screen(crI.BottomRight()) );
}

CRect ScreenMap::Image_to_Screen(const CRect crS, const CPoint offset)
{
    CRect r=Image_to_Screen(crS);
    r.OffsetRect(offset);
    return r;
}

void ScreenMap::ValidateZoom(double & x)
{
    if(x*crImage.Width()<16) x=16.0/crImage.Width();
    else if(x*crImage.Width()>4096) x=4096.0/crImage.Width();
    if(x*crImage.Height()<16) x=16.0/crImage.Height();
    else if(x*crImage.Height()>4096) x=4096.0/crImage.Height();
    int w_screen=4 * ( (2+(int)(0.5*x*crImage.Width()))/4 );
    int h_screen=4 * ( (2+(int)(0.5*x*crImage.Height()))/4 );
    crScreen.SetRect( crScreen.TopLeft().x, crScreen.TopLeft().y,
        crScreen.TopLeft().x+w_screen,
        crScreen.TopLeft().y+h_screen );
}

CSize ScreenMap::GetScreenCenteredSize()
{
    return CSize(crScreen.right+crScreen.left,crScreen.top+crScreen.bottom);
}

void ScreenMap::SetLeftTopScreenPoint(int x, int y)
{
    if(x<0) x=0;
    if(y<0) y=0;
    CSize offset=CPoint(x,y)-crScreen.TopLeft();
    crScreen.OffsetRect(offset);
}

bool ScreenMap::Screen_in_Image(const CPoint screenP, const int bound)
{
    CPoint p=Screen_to_Image(screenP);
    CRect ir=crImage; ir.DeflateRect(bound,bound);
    return (ir.PtInRect(p)==TRUE);
}

bool ScreenMap::Screen_in_Image(const CRect screenR, const int bound)
{
    return (Screen_in_Image(screenR.TopLeft(),bound) &&
        Screen_in_Image(screenR.BottomRight(),bound));
}

/*****
*
* Rotate screen map rectangles by 90 degrees.
*
*****/
void ScreenMap::Rotate(int degrees)
{
    if(degrees != 90 && degrees != 270) return; // nothing to do
    crScreen.SetRect(crScreen.left, crScreen.top,
        crScreen.left+crScreen.Height(),
        crScreen.top+crScreen.Width() );

    crImage.SetRect(crImage.left, crImage.top,
        crImage.left+crImage.Height(),
        crImage.top+crImage.Width() );
}

```

```

/*****
 *
 *   Serialize this screen map.
 *   Used as a part of image serialization
 *
 *****/
bool ScreenMap::SerializeScreenMap(FILE *fp, bool is_loading)
{
    int ix0,ix1,iy0,iy1,sx0,sx1,sy0,sy1;
    if(!is_loading)
    {
        ix0 = crImage.left;      ix1 = crImage.right;
        iy0 = crImage.top;       iy1 = crImage.bottom;
        sx0 = crScreen.left;     sx1 = crScreen.right;
        sy0 = crScreen.top;      sy1 = crScreen.bottom;
    }
    if(!::SerializeInteger(fp, ix0, is_loading)) return false;
    if(!::SerializeInteger(fp, ix1, is_loading)) return false;
    if(!::SerializeInteger(fp, iy0, is_loading)) return false;
    if(!::SerializeInteger(fp, iy1, is_loading)) return false;
    if(!::SerializeInteger(fp, sx0, is_loading)) return false;
    if(!::SerializeInteger(fp, sx1, is_loading)) return false;
    if(!::SerializeInteger(fp, sy0, is_loading)) return false;
    if(!::SerializeInteger(fp, sy1, is_loading)) return false;
    if(!::SerializeInteger(fp, ix1, is_loading)) return false;
    if(!::SerializeInteger(fp, iy0, is_loading)) return false;

    if(is_loading)
    {
        crImage.SetRect (ix0,iy0,ix1,iy1);
        crScreen.SetRect (sx0,sy0,sx1,sy1);
    }
    return true;
}

/*****
 *
 *   Place "crScreen" inside given "screen"
 *
 *****/
void ScreenMap::FitInsideScreenRect(CRect screen)
{
    double c = min(screen.Width()/(1.0+crScreen.Width()),
                   screen.Height()/(1.0+crScreen.Height()));
    int w = (int)max(4,c*crScreen.Width());
    int h = (int)max(4,c*crScreen.Height());
    int x = (screen.TopLeft().x+screen.BottomRight().x)/2;
    int y = (screen.TopLeft().y+screen.BottomRight().y)/2;
    crScreen = CRect(x-w/2,y-h/2,x+w/2,y+h/2);
}

/*****
 *
 *   Get current map zoom factor
 *
 *****/
double ScreenMap::GetZoom()
{
    return (crScreen.Width()+0.0001)/(crImage.Width()+0.0001);
}

```

```

#ifndef AFX_AEOPTIONS_DIALOG_H__INCLUDED_
#define AFX_AEOPTIONS_DIALOG_H__INCLUDED_

#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AEOptions_Dialog.h : header file
//

////////////////////
// AEOptions_Dialog dialog

class AEOptions_Dialog : public CDialog
{
// Construction
public:
    virtual int DoModal(ApplicationEntityList *AEarray);
    AEOptions_Dialog(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{AFX_DATA(AEOptions_Dialog)
    enum { IDD = IDD_DIALOG_AE_OPTIONS };
    BOOL m_useMoveToRetrieve;
    int m_Port;
    int m_PortServer;
    int m_Timeout;
    CString m_Comments;
    CIPAddressCtrl m_IP;
    CComboBox m_AEComboList;
    CString m_Title;
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(AEOptions_Dialog)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(AEOptions_Dialog)
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    afx_msg void OnCloseupComboAeList();
    afx_msg void OnAENew();
    afx_msg void OnAEDelete();
    afx_msg void OnAeClone();
    afx_msg void OnSelchangeComboAeList();
    //}AFX_MSG
    DECLARE_MESSAGE_MAP()

private:
    int m_ListIndex;
    ApplicationEntityList *m_AEarray;

    void ResetAeList(int new_selection=0);
    void UpdateAllFields(bool from_data_to_dialog=true);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_AEOPTIONS_DIALOG_H__INCLUDED_)

```

```
// AEOptions_Dialog.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "..\Resource.h"
#include "AEOptions_Dialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// AEOptions_Dialog dialog
```

```
AEOptions_Dialog::AEOptions_Dialog(CWnd* pParent /*=NULL*/)
: CDialog(AEOptions_Dialog::IDD, pParent)
```

```
{
    //{AFX_DATA_INIT(AEOptions_Dialog)
    m_useMoveToRetrieve = FALSE;
    m_Port = 0;
    m_PortServer = 0;
    m_Timeout = 0;
    m_Comments = _T("");
    m_Title = _T("");
    //}}AFX_DATA_INIT
    m_ListIndex=0;
}
```

```
void AEOptions_Dialog::DoDataExchange(CDataExchange* pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{AFX_DATA_MAP(AEOptions_Dialog)
    DDX_Check(pDX, IDC_CHECK_MOVETOGET, m_useMoveToRetrieve);
    DDX_Text(pDX, IDC_EDIT_PORT, m_Port);
    DDV_MinMaxInt(pDX, m_Port, 0, 70000);
    DDX_Text(pDX, IDC_EDIT_PORT_SERVER, m_PortServer);
    DDV_MinMaxInt(pDX, m_PortServer, 1, 70000);
    DDX_Text(pDX, IDC_EDIT_TIMEOUT, m_Timeout);
    DDV_MinMaxInt(pDX, m_Timeout, 0, 100000);
    DDX_Text(pDX, IDC_EDIT_COMMENTS, m_Comments);
    DDV_MaxChars(pDX, m_Comments, 63);
    DDX_Control(pDX, IDC_AE_IPADDRESS, m_IP);
    DDX_Control(pDX, IDC_COMBO_AE_LIST, m_AEComboList);
    DDX_Text(pDX, IDC_EDIT_TITLE, m_Title);
    DDV_MaxChars(pDX, m_Title, 16);
    //}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(AEOptions_Dialog, CDialog)
```

```
    //{AFX_MSG_MAP(AEOptions_Dialog)
    ON_CBN_CLOSEUP(IDC_COMBO_AE_LIST, OnCloseupComboAeList)
    ON_BN_CLICKED(ID_AE_NEW, OnAENew)
    ON_BN_CLICKED(ID_AE_DELETE, OnAEDelete)
    ON_BN_CLICKED(ID_AE_CLONE, OnAeClone)
    ON_CBN_SELCHANGE(IDC_COMBO_AE_LIST, OnSelchangeComboAeList)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// AEOptions_Dialog message handlers
```

```
/*
 *
 * Display modal dialog for AE setup
 *
 */
*/
```

```
int AEOptions_Dialog::DoModal(ApplicationEntityList *AEarray)
```

```
{
    if(!AEarray) return -1;
    m_AEarray = AEarray;
}
```

```

m_ListIndex=m_AEarray->GetCurrentIndex();
CDialog::DoModal();
return m_ListIndex;
}

/*****
*
*   Initialize all parameter fields
*
*****/
BOOL AEOptions_Dialog::OnInitDialog()
{
    CDialog::OnInitDialog();
    ResetAEList(m_ListIndex);
    GotoDlgCtrl(GetDlgItem(IDCANCEL));
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

/*****
*
*   Update all parameter fields
*
*****/
void AEOptions_Dialog::UpdateAllFields(bool from_data_to_dialog)
{
    int ind = m_AEComboList.GetCurSel();
    if(from_data_to_dialog)
    {
        if(ind>=(int)(m_AEarray->GetSize()) || ind<0) ind=m_ListIndex;
        else m_ListIndex=ind;
        ApplicationEntity* a = &(m_AEarray->Get(ind));
        m_IP.SetAddress(a->ae_IP1,a->ae_IP2,a->ae_IP3,a->ae_IP4);
        m_Title=CString(a->ae_Title);
        m_Port=a->ae_Port; m_PortServer=a->ae_PortServer;
        m_Timeout=a->ae_Timeout;
        m_Comments=CString(a->ae_Comments);
        m_useMoveToRetrieve=a->ae_useMoveAsGet;
        UpdateData(FALSE);
    }
    else
    {
        UpdateData(TRUE);
        if(ind>=(int)(m_AEarray->GetSize()) || ind<0) ind=m_ListIndex;
        else m_ListIndex=ind;
        CString location; m_AEComboList.GetLBText(ind,location);
        BYTE ip1, ip2, ip3, ip4; m_IP.GetAddress(ip1, ip2, ip3, ip4);
        m_AEarray->Get(ind).SetApplicationEntity ((char*)(LPCSTR)m_Title,
            ip1, ip2, ip3, ip4,m_Port,
            m_PortServer,m_Timeout,(char*)(LPCSTR)location,
            (char*)(LPCSTR)m_Comments,m_useMoveToRetrieve==TRUE);
    }
}

/*****
*
*   Combo List message handler
*
*****/
void AEOptions_Dialog::OnCloseupComboAeList()
{
    int n;
    int ind = m_AEComboList.GetCurSel();
    CString info;
    m_AEComboList.GetWindowText(info);
    if(ind != CB_ERR)
    {
        // If a valid choice was made from a listbox,
        // update all AE parameters
        m_ListIndex=ind;
        UpdateAllFields(true);
    }
    else if(info.IsEmpty() == TRUE)
    {

```



[illegible]

```
// MainFrm.cpp : implementation of the CMainFrame class
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "MainFrm.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CMainFrame
```

```
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
//{{AFX_MSG_MAP(CMainFrame)
ON_WM_CREATE()
ON_COMMAND(ID_VIEW_TOOLBAR, OnViewToolbar)
ON_UPDATE_COMMAND_UI(ID_VIEW_TOOLBAR, OnUpdateViewToolbar)
ON_WM_DROPFILES()
//}}AFX_MSG_MAP
ON_UPDATE_COMMAND_UI(ID_PROGRESS_STATUS, OnUpdateProgressStatus)
// Support dropdown toolbar buttons
ON_NOTIFY(TBN_DROPDOWN, AFX_IDW_TOOLBAR, OnToolbarDropDown)
// Global help commands
ON_COMMAND(ID_HELP_FINDER, CMDIFrameWnd::OnHelpFinder)
ON_COMMAND(ID_HELP, CMDIFrameWnd::OnHelp)
ON_COMMAND(ID_CONTEXT_HELP, CMDIFrameWnd::OnContextHelp)
ON_COMMAND(ID_DEFAULT_HELP, CMDIFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()
```

```
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_PROGRESS_STATUS,
    ID_INDICATOR_CAPS,
    //ID_INDICATOR_NUM,
    //ID_INDICATOR_SCRL,
    ID_DICTIONARY_STATUS
}
```

```
////////////////////////////////////
// CMainFrame construction/destruction
```

```
CMainFrame::CMainFrame()
{
    m_showToolbars=true;
    /* Dummy string to size progress indicator in the status bar */
    m_paneString=CString(' ',60);
}
```

```
CMainFrame::~CMainFrame()
{
}
```

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWnd::OnCreate(lpCreateStruct) == -1)    return -1;
    ShowWindow(SW_MAXIMIZE);

    // Create application toolbars
    if (!m_ToolBarBasic.CreateIE(this,32,32,
        IDR_TOOLBAR_BASIC))
    {
        return -1;        // fail to create
    }
    m_ToolBarBasic.AttachDropDown(ID_BUTTON_ROI, IDR_DCMTYPE, ID_BUTTON_SELECT_RECT);
    m_ToolBarBasic.AttachDropDown(ID_VIEW_FLIP, IDR_DCMTYPE, ID_VIEW_FLIP_VERTICAL);
    m_ToolBarBasic.AttachDropDown(ID_BUTTON_MEASURE, IDR_DCMTYPE,
        ID_BUTTON_MEASURE_RULER);
}
```

```

if (!m_ToolBarMultiframe.CreateIE(this,32,32,
    IDR_TOOLBAR_MULTIFRAME))
{
    return -1;    // fail to create
}
m_ToolBarMultiframe.AttachDropDown(ID_BROWSE_FRAME,IDR_DCMTYPE,
    ID_FRAMES_LAYOUT_STACK);
// Create application dialog bar
if (!m_wndDlgBar.Create(this, IDD_BAR_INFO,
    CBR_ALIGN_RIGHT, AFX_IDW_DIALOGBAR))
{
    TRACE0("Failed to create dialogbar\n");
    return -1;    // fail to create
}

// Create animated logo
if (!m_Animate.Create(WS_CHILD | WS_VISIBLE | ACS_AUTOPLAY,
    CRect(0,0,80,60), this, 0) ||
    !m_Animate.Open(IDR_AVI_DCM))
{
    TRACE0("Failed to create animation control\n");
    return -1;    // fail to create
}

// Create application ReBar
if (!m_ReBar.Create(this,0) ||
    !m_ReBar.AddBar(&m_Animate, NULL, NULL,RBBS_FIXEDBMP | RBBS_FIXEDSIZE) ||
    !m_ReBar.AddBar(&m_ToolBarBasic) ||
    !m_ReBar.AddBar(&m_ToolBarMultiframe) ||
    !m_ReBar.AddBar(&m_wndDlgBar))
{
    TRACE0("Failed to create rebar\n");
    return -1;    // fail to create
}

// Create application status bar
if (!m_StatusBar.Create(this) ||
    !m_StatusBar.SetIndicators(indicators,
    sizeof(indicators)/sizeof(UINT)))
{
    TRACE0("Failed to create status bar\n");
    return -1;    // fail to create
}

// Insert image counter into multiframe toolbar
m_ToolBarMultiframe.MakeCStatic(ID_FRAME_NUMBER);
ShowFrameNumber(-1);

// Size progress bar
CClientDC dc(this);
SIZE size=dc.GetTextExtent(m_paneString);
int index=m_StatusBar.CommandToIndex(ID_PROGRESS_STATUS);
m_StatusBar.SetPaneInfo(index,ID_PROGRESS_STATUS, SBPS_NORMAL, size.cx);

// Drag-and-drop file support
DragAcceptFiles();

return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // Size the main frame window to the screen size and center it
    return CMDIFrameWnd::PreCreateWindow(cs);
}

////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CMDIFrameWnd::AssertValid();
}

```

```

void CMainFrame::Dump(CDumpContext& dc) const
{
    CMDIFrameWnd::Dump(dc);
}

#endif //_DEBUG

////////////////////////////////////
// CMainFrame message handlers
void CMainFrame::OnUpdateProgressStatus(CCmdUI *pCmdUI)
{
    pCmdUI->Enable();
    pCmdUI->SetText(m_paneString);
}

/*****
*
*   Sets the number currently displayed in the frame edit toolbar
*
*****/
void CMainFrame::ShowFrameNumber(int n)
{
    CString st;
    if(n>0) st.Format("%d",n);
    else st.Format(" ");
    m_ToolBarMultiframe.SetInsertControlText(st);
}

/*****
*
*   Exit from application - clean up
*
*****/
BOOL CMainFrame::DestroyWindow()
{
    // Close main window
    return CMDIFrameWnd::DestroyWindow();
}

/*****
*
*   Show/hide all toolbars
*
*****/
void CMainFrame::OnViewToolbar()
{
    m_showToolbars = !m_showToolbars;
    m_ReBar.GetReBarCtrl().ShowBand(0, m_showToolbars);
    m_ReBar.GetReBarCtrl().ShowBand(1, m_showToolbars);
    m_ReBar.GetReBarCtrl().ShowBand(2, m_showToolbars);
}

void CMainFrame::OnUpdateViewToolbar(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck(m_showToolbars);
}

void CMainFrame::ShowMultiframeToolbar(bool show)
{
    m_ReBar.GetReBarCtrl().ShowBand(1, show);
}

/*****
*
*   Process dropdown toolbar buttons
*
*****/
void CMainFrame::OnToolbarDropDown(NMTOOLBAR *pnmtb, LRESULT *plr)
{
    m_ToolBarBasic.TrackDropDownMenu(pnmtb->iItem,this);
    m_ToolBarMultiframe.TrackDropDownMenu(pnmtb->iItem,this);
}

```

```

/*****
 *
 *   Drag-and-drop support
 *
 *****/
void CMainFrame::OnDropFiles(HDROP hDropInfo)
{
    // Find the number of files
    UINT nFiles = ::DragQueryFile(hDropInfo, (UINT)-1, NULL, 0);
    for(UINT iFile=0; iFile<nFiles; iFile++)
    {
        TCHAR szFileName[_MAX_PATH];
        ::DragQueryFile(hDropInfo, iFile, szFileName, _MAX_PATH);
        theApp.OpenDocumentFile(szFileName, true);
    }
    ::DragFinish(hDropInfo);
}

/*****
 *
 *   Switch logo animation on the menu bar
 *
 *****/
void CMainFrame::EnableLogoAnimation(bool enable)
{
    if(enable) m_Animate.Open(IDR_AVI_DCM);
    else      m_Animate.Stop();
}

/*****
 *
 *   Show dictionary availability on the status bar
 *
 *****/
void CMainFrame::SetDictionaryStatus(bool enabled)
{
    int index=m_StatusBar.CommandToIndex(ID_DICTIONARY_STATUS);
    CString stat = enabled ? "DICT ENABLED" : "DICT DISABLED";
    m_StatusBar.SetPaneText(index, stat);
}

```

```

// DICOMDocument.h: interface for the DICOMDocument class.
//
/////////////////////////////////////////////////////////////////

#ifdef AFX_DICOMDOCUMENT_H__INCLUDED_
#define AFX_DICOMDOCUMENT_H__INCLUDED_

#include "DICOMInfo.h" // Added by ClassView
#include "winmodules.h" // Added by ClassView
#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class DICOMDocument : public Array<ImageSeries> // Study
{
public:
    static const BYTE    FormatDICOMOriginal;
    static const BYTE    FormatDICOMModified;
    static const BYTE    FormatWINDOWSMultimedia;

    void                AddDICOMRecords(Array<DICOMRecord> &a);
    void                DisplayDICOMInfo();
    void                GetPixelSpacing(double &dx, double &dy);
    void                SetShowInfo(bool show);
    bool                GetShowInfo();
    bool                AddDDOFile(CString filename);
    bool                IsEmpty() { return m_DDO.IsEmpty(); }
    bool                SaveDICOM(CString filename="");
    bool                SaveDocument(CString fname, int format);
    bool                LoadFile(CString filename);
    bool                LoadDDO(DICOMDataObject &ddo, bool clone);
    int                GetNumberOfImages();
    int                GetCurrentImageIndex();
    CString             GetFilename() { return m_Filename; }
    CString             GetMRUAlias();
    Image*             GetImage(int n);
    inline ImageSeries* GetCurrentSeries()
    {
        return &m_ImageSeries;
        if(GetSize()<=0) return NULL; // empty
        if(m_CurrentSeries<0) m_CurrentSeries=0;
        if(m_CurrentSeries>=(int)GetSize()) m_CurrentSeries=GetUpperBound();
        return &(Get(m_CurrentSeries));
    };
    DICOMRecord*        GetDICOMRecordPtr() { return &m_FileRecord; };
    DICOMDocument();
    virtual ~DICOMDocument();

private:
    int                m_CurrentSeries;
    DICOMRecord        m_FileRecord;
    CString            m_Filename;
    PDU_Service        m_PDU;
    DICOMDataObject    m_DDO;
    DICOMInfo          m_InfoDialog;
    ImageSeries        m_ImageSeries;

    bool                InitializeDocument(DICOMDataObject& m_DDO,
                                           CString& m_Filename);
};

#endif // !defined(AFX_DICOMDOCUMENT_H__INCLUDED_)

```

```
// DICOMDocument.cpp: implementation of the DICOMDocument class.
```

```
//
```

```
////////////////////////////////////
```

```
#include "stdafx.h"
#include "../DCM.h"
#include "DICOMDocument.h"
```

```
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif
```

```
////////////////////////////////////
// Construction/Destruction
////////////////////////////////////
```

```
const BYTE DICOMDocument::FormatDICOMOriginal = 0;
const BYTE DICOMDocument::FormatDICOMModified = 1;
const BYTE DICOMDocument::FormatWINDOWSMultimedia = 2;
DICOMDocument::DICOMDocument() : m_InfoDialog(&theApp.app_RTC),
                                theApp.app_DirectoryTmp)
```

```
{
    if(!theApp.app_RTC.IsEmpty())
    {
        m_PDU.AttachRTC(&theApp.app_RTC), FALSE;
    }
    m_CurrentSeries = -1;
}
```

```
DICOMDocument::~DICOMDocument()
```

```
{
    //*****
    * Load DICOM Data from
    * 1. A file
    * 2. Another DDO
    ******/
```

```
bool DICOMDocument::LoadFile(CString filename)
```

```
{
    filename.TrimRight();    filename.TrimLeft();
    if(filename=="")
```

```
{
```

```
    AfxMessageBox("Cannot load: file name is empty",
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
```

```
// Store filename
m_Filename=filename;
// Load DDO, completely, using m_PDU for RTC
m_DDO.Reset();
if(!m_DDO.LoadFromFile((char*)(LPCSTR)(filename),false,&m_PDU))
```

```
{
    AfxMessageBox("Cannot read DICOM from file\n"+filename,MB_OK|MB_ICONEXCLAMATION);
    return false;
}
```

```
// Initialize data
if(!InitializeDocument(m_DDO, m_Filename)) return false;

// OK
return true;
}
```

```
bool DICOMDocument::LoadDDO(DICOMDataObject &ddo, bool clone)
```

```
{
    VR * vr;
    // Set m_DDO
    m_DDO.Reset();
    if (clone)
    {
```

```

    if(!m_DDO.CloneFrom( ))
    {
        AfxMessageBox("Cannot load DICOM",MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
else
{
    while(vr=ddo.Pop()) m_DDO.Push(vr);
}
// Set filename
char ff[65];
m_DDO.SuggestFileName(ff,65);
m_Filename = theApp.app_DirectoryTmp+"/"+CString(ff);

// Classify VRs
theApp.app_RTC.RunTimeClass(&m_DDO);

// Save m_DDO copy into m_Filename for consistency
if( !m_DDO.SaveIntoFile((char*)(LPCSTR) m_Filename),true,&m_PDU) )
{
    AfxMessageBox("Cannot save DICOM in file \n"+m_Filename,
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
// Initialize data
if(!InitializeDocument(m_DDO, m_Filename)) return false;
return true;
}

/*****
*
* Initialize all document data from m_DDO and m_Filename
*
*****/
bool DICOMDocument::InitializeDocument(DICOMDataObject& m_DDO,
    CString& m_Filename)
{
    // Do we have anything inside the DICOM object ?
    if(m_DDO.IsEmpty())
    {
        AfxMessageBox("Empty or invalid DICOM object", MB_ICONEXCLAMATION|MB_OK);
        return false;
    }
    // Display, parse and hide DICOM Info
    //if(!m_InfoDialog.PopInfo(m_DDO, m_Filename)) return false;
    // Set file record
    m_FileRecord.SetRecord(m_DDO, (char*)(LPCSTR)m_Filename);
    // Parse image data into bitmaps
    m_ImageSeries.ReadDO(m_DDO, (char*)(LPCSTR)m_Filename);
    //Array<ImageSeries>::Add(m_ImageSeries);
    //m_CurrentSeries = Array<ImageSeries>::GetUpperBound();

    // If no images found, display info dialog
    if(GetNumberOfImages(<1) m_InfoDialog.DoModeless(m_DDO,m_Filename);
    return true;
}

/*****
*
* Save DICOM Data into a file
*
*****/
bool DICOMDocument::SaveDICOM(CString filename /* ="" */)
{
    if(filename=="") filename=m_Filename;
    if(filename=="")
    {
        AfxMessageBox("Empty filename, cannot save",MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    // Save copy DO
    DICOMDataObject do_tmp;
    if(!do_tmp.CloneFrom(&m_DDO))

```



```

{
    AfxMessageBox("Cannot save DICOM data",MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!GetCurrentSeries()->WriteDO(do_tmp))
{
    AfxMessageBox("Cannot save image data",MB_OK|MB_ICONEXCLAMATION);
    return false;
}
if(!do_tmp.SaveIntoFile((char*)(LPCSTR)(filename),false))
{
    AfxMessageBox("Cannot save DICOM in file \n"+filename,
        MB_OK|MB_ICONEXCLAMATION);
    return false;
}
return true;
}

```

```

/*****
*
*   Display DICOM Object
*
*****/

```

```

void DICOMDocument::DisplayDICOMInfo()
{
    m_InfoDialog.DoModal(m_DDO,m_Filename);
}

```

```

/*****
*
*   Navigating in image series
*
*****/

```

```

Image* DICOMDocument::GetImage(int n)
{
    ImageSeries* pS = GetCurrentSeries();
    if(!pS) return NULL;
    else return pS->GetImage(n);
}

int DICOMDocument::GetNumberOfImages()
{
    ImageSeries* pS = GetCurrentSeries();
    if(!pS) return 0;
    else return pS->GetUpperBound()+1;
}

int DICOMDocument::GetCurrentImageIndex()
{
    ImageSeries* pS = GetCurrentSeries();
    if(!pS) return -1;
    else return pS->GetCurrentImageIndex();
}

```

```

/*****
*
*   Save the entire document in specified format
*
*****/

```

```

bool DICOMDocument::SaveDocument(CString fname, int format)
{
    bool success=true;
    CString err("Error saving the document");
    // Clean proposed file name
    fname.TrimLeft();    fname.TrimRight();
    if(fname=="")
    {
        AfxMessageBox("Cannot save: file name is empty",
            MB_OK|MB_ICONEXCLAMATION);
        return false;
    }

    // Save in requested format
    if(format==FormatDICOMOriginal) // Save unchanged DICOM
    {

```

```

        if(fname!=m_Filename)
        {
            bool success=(CopyFile(m_Filename,fname,FALSE)!=0);
        }
    }
    else if(format==FormatDICOMModified)        // Save modified DICOM
    {
        return SaveDICOM(fname);
    }
    else if(format==FormatWINDOWSMultimedia)    // Save DICOM as multimedia directory
    {
        // Create new directory
        if(!::CreateAndSetCurrentDirectory((char*)(LPCSTR)fname))
        {
            AfxMessageBox("Cannot access directory\n"+fname,
                MB_OK|MB_ICONEXCLAMATION);
            return false;
        }
        // Save all bitmaps
        success = GetCurrentSeries()->SaveAsImageFiles(fname);
        // Save DICOM demographics
        char info[_MAX_PATH];
        sprintf(info,"%s\\info.txt",fname);
        success = success && m_FileRecord.WriteIntoTextFile(info);

        // Save sound data
        /*
        if(m_SoundFileName.GetLength()>3)
        {
            try
            {
                CFile::Rename(m_SoundFileName, fname+"\\voice.wav");
            }
            catch (CFileException *e)
            {
                err.Format("Error saving sound file into %s",fname+fname+"voice.wav");
                AfxMessageBox(err);
                e->Delete();
                return FALSE;
            }
        }
        */
    }
    else
    {
        err="Invalid format";    success=false;
    }
    if(!success)    AfxMessageBox(err,MB_OK|MB_ICONEXCLAMATION);
    Beep(500,100);
    return success;
}

/*****
 *
 *   Get physical spacing between image pixels
 *
 *****/
void DICOMDocument::GetPixelSpacing(double &dx, double &dy)
{
    GetCurrentSeries()->GetPixelSpacing(dx, dy);
}

/*****
 *
 *   Show image info on the images
 *
 *****/
void DICOMDocument::SetShowInfo(bool show)
{
    GetCurrentSeries()->SetShowSeriesImageInfo(show);
}
bool DICOMDocument::GetShowInfo()
{
    return GetCurrentSeries()->GetShowSeriesImageInfo();
}

```

```
}

/*****
 *
 *   Create an alias name for the MRU files list
 *
 *****/
CString DICOMDocument::GetMRUAlias()
{
    CString s;
    s.Format("%s, %s", m_FileRecord.GetPatientName(), m_FileRecord.GetPatientID());
    return s;
}

/*****
 *
 *   Include new series files
 *
 *****/
bool DICOMDocument::AddDDOFile(CString filename)
{
    ImageSeries* pS = GetCurrentSeries();
    if(pS) return pS->AddDDOFile(filename);
    else return false;
}

void DICOMDocument::AddDICOMRecords(Array<DICOMRecord> &a)
{
    if(a.GetSize()<=0) return;
    ImageSeries* pS = GetCurrentSeries();
    if(!pS)
    {
        LoadFile(a[0].GetFileName());
        a.RemoveAt(0);
        AddDICOMRecords(a); // recursion
        return;
    }
    else pS->AddDICOMRecords(a);
}
```

```

#if !defined(AFX_DICOMINFO_H_INCLUDED_)
#define AFX_DICOMINFO_H_INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "../resource.h"
#include <dicom.hpp>

////////////////////////////////////
// DICOMInfo dialog

class DICOMInfo : public CDialog, public DICOMView
{
// Construction
public:
    void SetFilename(CString filename);
    void DoModeless();
    void DoModeless(DICOMObject& dob, CString filename);
    void Load(const char* str);
    void Load(DICOMObject &DO);
    bool PopInfo(DICOMObject& dob, CString& filename);
    bool LoadFile(char* filename);
    DICOMInfo(RTC* rtc, CString temp_dir, CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{AFX_DATA(DICOMInfo)
    enum { IDD = IDD_DIALOG_DICOMINFO };
    //}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(DICOMInfo)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{AFX_MSG(DICOMInfo)
    afx_msg void OnOK();
    afx_msg void OnCancel();
    afx_msg void OnDropFiles(HDROP hDropInfo);
    virtual BOOL OnInitDialog();
    //}AFX_MSG
    afx_msg void OnClose();
    DECLARE_MESSAGE_MAP()

private:
    CString m_Filename, m_BackupFile, m_TemporaryDirectory;
    const CString m_DropFile;
    CListCtrl m_List;

    void SaveToFile();
    void ResetBackup();
    bool OpenFromFile();
};

//{AFX_INSERT_LOCATION}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DICOMINFO_H_INCLUDED_)

```

```

// dicominfo.cpp : implement n file
//

#include "stdafx.h"
#include "dicominfo.h"
#include <io.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// DICOMInfo dialog

DICOMInfo::DICOMInfo(RTC* rtc, CString temp_dir, CWnd* pParent /*=NULL*/)
: CDialog(DICOMInfo::IDD, pParent), m_DropFile("")
{
    //{{AFX_DATA_INIT(DICOMInfo)
    m_Filename = _T("");
    //}}AFX_DATA_INIT
    // Set RTC
    if(!rtc->IsEmpty()) AttachRTC(rtc);
    // Set temporary directory
    m_TemporaryDirectory=temp_dir;
    m_TemporaryDirectory.Replace('/', '\\');
    m_TemporaryDirectory.TrimRight(" \\");
    // Set backup file name
    m_BackupFile="";
}

void DICOMInfo::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DICOMInfo)
    DDX_Control(pDX, IDC_LIST, m_List);
    DDX_Text(pDX, IDC_FILENAME, m_Filename);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DICOMInfo, CDialog)
    //{{AFX_MSG_MAP(DICOMInfo)
    ON_WM_DROPFILES()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// DICOMInfo message handlers

/*****
*
*   Overriden virtuals from DICOMView
*
*****/
void DICOMInfo::Load(const char *str)
{
    int n,m;
    CString cs=CString(str);
    cs.TrimRight(); cs.TrimLeft();
    if(cs=="") return;
    // Parse VR string
    CString tag; n=cs.Find(",");
    if(n<0) tag="";
    else { tag=cs.Left(n+1); cs=cs.Mid(n+2); }
    tag.TrimRight();

    CString length; n=cs.Find("bytes,");
    if(n<0) length="";
    else { length=cs.Left(n+5); cs=cs.Mid(n+6); }
    length.TrimLeft(); length.TrimRight();
}

```

```

CString vr;
n=cs.Find("VR=");
if(n<0) vr="";
else {   vr=cs.Mid(n+4,2);   cs=cs.Mid(n+7); }
vr.TrimLeft();  vr.TrimRight();

CString value, description;
n=cs.Find("<"); m=cs.Find("[",max(n,0));
if(n<0) value="";
else
{
    if(m>n)
    {
        value=cs.Mid(n,m-n);   description=cs.Mid(m+2);
        description.TrimRight("]");
    }
    else
    {
        value=cs.Mid(n);       description="";
    }
}
value.TrimLeft();   value.TrimRight();
description.TrimLeft(); description.TrimRight();

UINT nItem=m_List.InsertItem(m_List.GetItemCount(),tag);
m_List.SetItem(nItem,1,LVIF_TEXT,length,0,0,0,0);
m_List.SetItem(nItem,2,LVIF_TEXT,vr,0,0,0,0);
m_List.SetItem(nItem,3,LVIF_TEXT,value,0,0,0,0);
m_List.SetItem(nItem,4,LVIF_TEXT,description,0,0,0,0);
}

void DicomInfo::Load(DICOMObject &DO)
{
    bool top_level=(m_SequenceLevel==0);
    if(top_level)
    {
        GetDlgItem(IDC_EDITNUM)->SetWindowText("No elements found");
        m_List.DeleteAllItems();
        BeginWaitCursor();
        if(OpenFromFile()) return;
    }
    DicomView::Load(DO);
    if(top_level)
    {
        CString num;   num.Format("%d",m_numElements);
        GetDlgItem(IDC_EDITNUM)->SetWindowText(num);
        for(int i=0; i<=4; i++)
        {
            m_List.SetColumnWidth(i,LVSCW_AUTOSIZE);
        }
        EndWaitCursor();
        SaveToFile();
    }
}

bool DicomInfo::LoadFile(char *filename)
{
    BeginWaitCursor();
    m_List.DeleteAllItems();
    bool success = true;
    CString fn(filename);   fn.TrimRight();
    if(fn==m_Filename && OpenFromFile()) success=true;
    else success=DicomView::LoadFile(filename);
    if(!success) GetDlgItem(IDC_EDITNUM)->SetWindowText(
        "Cannot load this file");

    SetFilename(fn);
    EndWaitCursor();
    return success;
}

```

```

/*****
*
*   Modeless display and distruction
*
*****/

```

```

void DICOMInfo::DoModeless()
{
    if(this->GetSafeHwnd()) return;
    Create(IDD_DIALOG_DICOMINFO,NULL);
    // Always display on top
    SetWindowPos(&wndTopMost, 0,0,0,0,SWP_NOSIZE | SWP_SHOWWINDOW );
    SetWindowText("DICOM Header");
    ShowWindow(SW_SHOWNORMAL);
}

void DICOMInfo::DoModeless(DICOMObject &dob, CString filename)
{
    if(this->GetSafeHwnd()) return;
    DoModeless();
    SetFilename(filename);
    Load(dob);
}

void DICOMInfo::OnOK() { DestroyWindow(); }
void DICOMInfo::OnCancel() { DestroyWindow(); }
void DICOMInfo::OnClose() { DestroyWindow(); }
bool DICOMInfo::PopInfo(DICOMObject &dob, CString &filename)
{
    DoModeless(dob, filename);
    bool non_empty = (m_numElements>0);
    OnClose();
    if(!non_empty)
    {
        AfxMessageBox("Invalid DICOM format", MB_OK|MB_ICONEXCLAMATION);
    }
    return non_empty;
}

/*****
 *
 * Dragging files
 *
 *****/
void DICOMInfo::OnDropFiles(HDROP hDropInfo)
{
    // Find the number of files
    UINT nFiles = ::DragQueryFile(hDropInfo, (UINT)-1, NULL, 0);
    if(nFiles>1)
    {
        AfxMessageBox("Cannot display multiple files."
            "\nSelect one file and try again.",
            MB_OK|MB_ICONEXCLAMATION);
        return;
    }
    TCHAR szFileName[256];
    ::DragQueryFile(hDropInfo,0,szFileName,256);
    CString temp=m_BackupFile; m_BackupFile=m_DropFile;
    LoadFile(szFileName);
    m_BackupFile=temp;
    ::DragFinish(hDropInfo);
}

/*****
 *
 * Set filename
 *
 *****/
void DICOMInfo::SetFilename(CString filename)
{
    m_Filename=filename; m_Filename.TrimRight();
    UpdateData(FALSE);
}

BOOL DICOMInfo::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Element list columns
    m_List.InsertColumn(0,"Element Tag", LVCFMT_CENTER,60,0);

```

```

        m_List.InsertColumn(1,"Label", LVCFMT_CENTER,80,0);
        m_List.InsertColumn(2,"Value", LVCFMT_CENTER,80,0);
        m_List.InsertColumn(3,"Value", LVCFMT_LEFT,100,0);
        m_List.InsertColumn(4,"Description", LVCFMT_LEFT,80,0);
        // Element list: Force entire row selection
        m_List.SendMessage( LVM_SETEXTENDEDLISTVIEWSTYLE, 0,LVS_EX_FULLROWSELECT );
        return TRUE; // return TRUE unless you set the focus to a control
                    // EXCEPTION: OCX Property Pages should return FALSE
    }

/*****
*
*   Persistent Storage
*
*****/
void DICOMInfo::SaveToFile()
{
    if(m_BackupFile==m_DropFile)    return;
    if(m_BackupFile=="")    // was not initialized
    {
        char curdir[MAX_PATH+1];
        GetCurrentDirectory(MAX_PATH,curdir);
        SetCurrentDirectory(m_TemporaryDirectory);
        char name_template[13] = "_info_XXXXXX";
        bool success = (mktemp(name_template) != NULL);
        SetCurrentDirectory(curdir);
        if(!success)    return;
        m_BackupFile=m_TemporaryDirectory+"\\ "+CString(name_template);
    }
    int i, n;
    CString tag, length, vr, value, description;
    try
    {
        CFile db_file(m_BackupFile, CFile::modeCreate | CFile::modeWrite);
        CArchive ar(&db_file, CArchive::store);
        n=m_List.GetItemCount();
        ar<<n;
        for(i=0; i<n; i++)
        {
            tag=m_List.GetItemText(i,0);
            length=m_List.GetItemText(i,1);
            vr=m_List.GetItemText(i,2);
            value=m_List.GetItemText(i,3);
            description=m_List.GetItemText(i,4);
            ar<<tag<<length<<vr<<value<<description;
        }
        ar.Close();
    }
    catch(CException* e)
    {
        e->Delete();
        ResetBackup();
    }
}

bool DICOMInfo::OpenFromFile()
{
    if(m_BackupFile==" " || m_BackupFile==m_DropFile)    return false;
    int i, n;
    UINT nItem;
    CString tag, length, vr, value, description;
    try
    {
        CFile db_file(m_BackupFile, CFile::modeRead);
        CArchive ar(&db_file, CArchive::load);
        ar>>n;
        m_List.DeleteAllItems();
        for(i=0; i<n; i++)
        {
            ar>>tag>>length>>vr>>value>>description;
            nItem=m_List.InsertItem(m_List.GetItemCount(),tag);
            m_List.SetItem(nItem,1,LVIF_TEXT,length,0,0,0,0);
            m_List.SetItem(nItem,2,LVIF_TEXT,vr,0,0,0,0);
            m_List.SetItem(nItem,3,LVIF_TEXT,value,0,0,0,0);

```



```

        m_List.SetItem(n, 4, LVIF_TEXT, description, 0, 0, 0, 0)
    }
    ar.Close();
    m_numElements=n;
    CString num;    num.Format("%d",m_numElements);
    GetDlgItem(IDC_EDITNUM) ->SetWindowText(num);
    for(i=0; i<=4; i++)
    {
        m_List.SetColumnWidth(i, LVSCW_AUTOSIZE);
    }
    return true;
}
catch(CException* e)
{
    e->Delete();
    ResetBackup();
    return false;
}
}

```

```

/*****

```

```

*   Reset the backup file

```

```

*****/

```

```

void DICOMInfo::ResetBackup()

```

```

{
    DeleteFile(m_BackupFile);
    m_BackupFile="";
}

```

```

// Image.h: interface for the Image class.
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_IMAGE_H_INCLUDED_
#define AFX_IMAGE_H_INCLUDED_

#include "ScreenMap.h"
#include "Palette.h"
#include "../StdAfx.h" // Added by ClassView

#define COMPARE_CORRELATION 0
#define COMPARE_ABSOLUTE 1

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
class Image : public COObject
{
public:
    bool m_UpdateBitmap;
    bool m_RGB;
    BYTE m_EdgeThreshold;
    static const BYTE
        DisplayNormal, DisplayHidden, DisplaySelected;
    long m_maxPixelValue;
    Palette* m_pPal;
    ScreenMap m_ScreenMap;

    void SetImageData(DICOMRecord* dr, Point2D* pspace, int* index);
    void SetImageRectZoom(double zoom);
    void SetImageRectOffset(double dx, double dy);
    void SetDisplayStatus(BYTE stat);
    void LinkToPalette(Palette* pPal);
    void GetSubimage(Image* sub, const int xleft, const int ytop);
    void Palette_to_Pixels();
    void ResetPixels(bool current_to_safe);
    void SetBrightness(int b);
    void TR_Flip(bool horizontal);
    void SetShowImageInfo(bool show) { m_ShowImageInfo = show; };
    void GetStatistics(long &avg, long &sigma2, long &count,
        int x0, int x1, int y0, int y1);
    void GetStatistics(long &avg, long &sigma2, long &count)
    {
        GetStatistics(avg, sigma2, count, 0, m_Width, 0, m_Height);
    };
    inline void SetPixel(unsigned int x, unsigned int y, long p);
    inline void SetPixelFromLum(unsigned int x, unsigned int y, long p);
    inline void SetPixel(unsigned long n, long p);
    bool ContainsScreenPoint(CPoint& p);
    bool GetShowImageInfo() { return m_ShowImageInfo; };
    bool CloneFrom(Image* img);
    bool WriteToFile(CString fname);
    bool SetPalette(long* color_map, long color_map_size, bool show_progress=true);
    bool SetPalette(int px, int py, int pxmax, int pymax);
    bool CreateImage(int xwidth, int xheight,
        int bytes_per_pixel, Palette* pPal=NULL);
    bool TR_SetUniformLuminance(CRect r, bool show_progress=true);
    bool TR_SetUniformLuminance(bool show_progress=true);
    bool TR_PixelNeighborhood(char mask_type, bool show_progress);
    bool TR_PixelExp();
    bool TR_PixelLog();
    bool TR_GammaCorrection(double gamma);
    bool TR_HistStretch(BYTE percent, bool show_progress=true);
    bool TR_HistStretch(int amin, int amax, int bmin, int bmax,
        bool show_progress=true);
    bool TR_HistEqualize(bool show_progress=true);
    bool TR_Negate();
    bool TR_Rotate(int degrees);
    bool DisplayDIB(CDC *pDC, CRect crectScreenArea, CRect crectImageArea,
        CPoint pScroll=CPoint(0,0), bool optimize=true);
    bool DisplayDIB(CDC *pDC, CPoint pScroll);
    bool DisplayDIB(CDC *pDC);

```

```

BYTE*      GetPixelByte(NT32& data_size);
inline BYTE GetDisplayStatus() { return m_DisplayStatus; }
int        CompareToDICOMRecord(DICOMRecord & dr, BYTE lev);
inline int  GetHeight() { return m_Height; };
inline int  GetWidth() { return m_Width; };
inline int  GetBytesPerPixel() { return m_Bytes_per_Pixel; }
inline int  GetBytesInRow() { return m_Height*m_Bytes_per_Pixel; }
long        TR_Fractal(const int x, const int y);
inline long GetPixel(unsigned int x, unsigned int y);
inline long GetPixel(unsigned long n);
inline long GetLuminance(int x, int y);
inline long GetSmoothedLuminance(int x, int y);
double      Compare(int comp_type, CPoint topleft, long **pattern,
                    int pat_w, int pat_h, long pat_avr);

inline double
    GetZoom() { return m_ScreenMap.GetZoom(); };

CPoint      TR_FindSimilarRegion(const CRect r0);
Image(bool undo=true);
~Image();

private:
    bool        m_ReleasePalette;
    bool        m_DisplayFailed;
    bool        m_ShowImageInfo;
    BYTE        m_DisplayStatus;
    BYTE*       m_Pixels;
    unsigned int m_numColors;
    int         m_Index;
    int         m_Height, m_Width;
    int         m_Bytes_per_Pixel;
    long        m_numPixelBytes;
    unsigned long m_numPixels;
    static unsigned long
        m_UndoFileCount;
    Point2D     m_PixelSpacing;
    CString     m_UndoFile;
    HBITMAP     m_Bitmap;
    CStatusBar* m_MainStatusBar;
    BITMAPFILEHEADER m_bmpFHeader;
    LPBITMAPINFO m_bmpInfo;
    DICOMRecord m_DICOMRecord;

    void        DisplayImageInfo(CDC* pDC);
    void        FormatImageInfo(CString& info);
    void        Get_Pixel_minmax(long& pmin, long& pmax);
    void        DeleteImageData();
    bool        SerializeImage(FILE* fp, bool is_loading);
    inline unsigned long
        MapPixel(unsigned int x, unsigned int y);
    long        TR_DeNoise(const int x, const int y);
    long        TR_Sharp(const int x, const int y);
    long        TR_Smooth(const int x, const int y);
    long        TR_Sobel(const int x, const int y);
    HBITMAP     DIB_to_DDB(CDC *pDC);
};

/*****
 *
 *   Get pixel functions
 *
 *****/
inline long Image::GetPixel(unsigned int x, unsigned int y)
{
    return GetPixel(MapPixel(x,y));
}

```

```

}
inline long Image::GetPixel(unsigned long n)
{
    if(n>=m_numPixels) n=m_numPixels-1;
    n *= m_Bytes_per_Pixel; // array index
    // Color ?
    if(m_RGB) return RGB(m_Pixels[n+2],m_Pixels[n+1],m_Pixels[n]);
    // Grayscale
    switch(m_Bytes_per_Pixel)
    {
        case 1: return m_Pixels[n];
        case 2: return ((UINT16*)m_Pixels)[n];
        case 3: return (m_Pixels[n+1] + (((long)m_Pixels[n+1])<<8)
                        + (((long)m_Pixels[n+2])<<16) );
        case 4: return ((UINT32*)m_Pixels)[n];
    }
    return m_Pixels[n];
}

/*****
*
*   Set pixel functions
*
*****/
inline void Image::SetPixel(unsigned int x, unsigned int y, long p)
{
    m_UpdateBitmap=true;
    SetPixel(MapPixel(x,y),p);
}
inline void Image::SetPixelFromLum(unsigned int x, unsigned int y, long p)
{
    m_UpdateBitmap=true;
    long n=MapPixel(x,y);
    if(m_RGB) // color image
    {
        m_Pixels[n] = (BYTE)p;
        m_Pixels[n+1]=(BYTE)p;
        m_Pixels[n+2]=(BYTE)p;
        return;
    }
    else SetPixel(n,p);
}
inline void Image::SetPixel(unsigned long n, long p)
{
    m_UpdateBitmap=true;
    if(n>=m_numPixels) n=m_numPixels-1;
    // Color ?
    if(m_RGB)
    {
        m_Pixels[n]=GetBValue(p);
        m_Pixels[n+1]=GetGValue(p);
        m_Pixels[n+2]=GetRValue(p);
        return;
    }
    // Grayscale
    if(p>m_maxPixelValue) p=m_maxPixelValue;
    else if(p<0) p=0;
    switch(m_Bytes_per_Pixel)
    {
        case 1: m_Pixels[n]=(BYTE)p;
                break;
        case 2: ((UINT16*)m_Pixels)[n] = (UINT16)p;
                break;
        case 3: m_Pixels[n+2]=(BYTE) (p>>16);
                m_Pixels[n+1]=(BYTE) ((p>>8)&255);
                m_Pixels[n]=(BYTE) (p&255);
                break;
        case 4: ((UINT32*)m_Pixels)[n] = (UINT32)p;
                break;
    }
    return;
}

```

```

/*****
 *
 * Returns luminance (brightness) of pixel (x,y)
 *
 *****/
inline long Image::GetLuminance(int x, int y)
{
    if(x<=0) x=0;          if(y<0) y=0;
    if(m_RGB)
    {
        long i=MapPixel(x,y);
        return (long)((m_Pixels[i]+m_Pixels[i+1]+m_Pixels[i+2])/3);
    }
    else return GetPixel(x,y);
}

inline long Image::GetSmoothedLuminance(int x, int y)
{
    if(x<0) x=0;
    if(y<0) y=0;
    long p=TR_Smooth(x,y);
    if(m_RGB)
    {
        return (long)((GetRValue(p)+GetGValue(p)+GetBValue(p))/3);
    }
    else return p;
}

#endif // !defined(AFX_IMAGE_H_INCLUDED_)

```

```

// Image.cpp: implementation of the Image class.
//
/////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "..\DCM.h"
#include "Image.h"
#include "..\FindRegion.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#define new DEBUG_NEW
#endif

/////////////////////////////////////////////////////////////////
// Construction/Destruction
/////////////////////////////////////////////////////////////////
const BYTE Image::DisplayNormal = 0;
const BYTE Image::DisplayHidden = 1;
const BYTE Image::DisplaySelected = 2;
unsigned long Image::m_UndoFileCount=0;
Image::Image(bool undo /*=true*/)
{
    m_Pixels=NULL;
    m_bmpInfo=NULL;
    m_Bitmap=NULL;
    m_pPal=NULL;
    m_ReleasePalette = true;
    if(undo) m_UndoFile = "+";
    else m_UndoFile = "";
    m_ShowImageInfo = false;
    m_DisplayStatus = DisplayNormal;
    m_Index=0;
}

Image::~Image()
{
    DeleteImageData();
    if((m_UndoFile != "") && (m_UndoFile != "+")) DeleteFile(m_UndoFile);
}

void Image::DeleteImageData()
{
    if(m_Pixels) { delete [] m_Pixels; m_Pixels=NULL; }
    if(m_bmpInfo) { delete [] m_bmpInfo; m_bmpInfo=NULL; }
    if(m_Bitmap) DeleteObject(m_Bitmap);
    if(m_ReleasePalette)
    {
        if(m_pPal) delete m_pPal;
        m_pPal=0;
    }
}

void Image::SetImageData(DICOMRecord *dr, Point2D *pspace, int *index)
{
    if(dr) m_DICOMRecord = (*dr);
    if(pspace) m_PixelSpacing = (*pspace);
    if(index) m_Index = (*index);
}

/*****
*
* Initialize image pixel arrays. Always call after image construction
*
*****/
bool Image::CreateImage(int xwidth, int xheight, int bytes_per_pixel,
                        Palette* pPal /*=NULL*/)
{
    // Clear if anything existed
    DeleteImageData();
    // Find number of bytes per pixel with the current display
    if(bytes_per_pixel>=1 && bytes_per_pixel<=3)
    {

```

```

        m_Bytes_per_Pixel= bytes_per_pixel;
    }
    else return false; // out of memory
    int device_bpp = (theApp.app_Metheus ? 2 : 1); // bytes per pixel available
    if (device_bpp<m_Bytes_per_Pixel) m_Bytes_per_Pixel=device_bpp;
    m_RGB=false; // no color images so far

    // Set image parameters
    if(xwidth>8)    m_Width=8*(xwidth/8);        else m_Width=8;
    if(xheight>8)   m_Height=8*(xheight/8);       else m_Height=8;

    if(m_Bytes_per_Pixel==1)    m_numColors=256;
    else                        m_numColors=0;

    m_numPixels=((long)m_Width)*m_Height; // total number of pixels
    m_numPixelBytes=m_numPixels*m_Bytes_per_Pixel;
    if(!m_RGB) m_maxPixelValue=__min(4095, (1<<(8*m_Bytes_per_Pixel))-1);
    else      m_maxPixelValue=255;
    m_EdgeThreshold=max(1,m_maxPixelValue/20); // 5% of the total scale

    // Set bitmap infoheader
    m_bmpInfo=(LPBITMAPINFO)new BYTE[sizeof(BITMAPINFOHEADER)
        +m_numColors*sizeof(RGBQUAD)];
    if(!m_bmpInfo) return false; // out of memory for bitmapinfo
    m_bmpInfo->bmiHeader.biSize=sizeof(BITMAPINFOHEADER); //fixed
    m_bmpInfo->bmiHeader.biWidth=m_Width;
    m_bmpInfo->bmiHeader.biHeight=m_Height;
    m_bmpInfo->bmiHeader.biPlanes=1; //fixed
    m_bmpInfo->bmiHeader.biBitCount=8*m_Bytes_per_Pixel;
    m_bmpInfo->bmiHeader.biCompression=BI_RGB; //fixed
    m_bmpInfo->bmiHeader.biSizeImage=0; //fixed
    m_bmpInfo->bmiHeader.biXPelsPerMeter=0; //fixed
    m_bmpInfo->bmiHeader.biYPelsPerMeter=0; //fixed
    m_bmpInfo->bmiHeader.biClrUsed=0;
    m_bmpInfo->bmiHeader.biClrImportant=0; //fixed, normally 0
    // Set bitmap palette
    for(unsigned int j=0; j<m_numColors; j++)
    {
        m_bmpInfo->bmiColors[j].rgbRed=j;
        m_bmpInfo->bmiColors[j].rgbGreen=j;
        m_bmpInfo->bmiColors[j].rgbBlue=j;
        m_bmpInfo->bmiColors[j].rgbReserved=0;
    }

    // Set default screen map
    m_ScreenMap.Initialize(CRect(0,0,m_Width,m_Height));
    m_ScreenMap.crScreen.OffsetRect(50,0);

    // Set default bitmap fileheader (used only for file I/O)
    m_bmpFHeader.bfType=('M'<<8)|'B'; //fixed
    m_bmpFHeader.bfSize=sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFO)
        +m_Width*m_Height*m_numColors;
    m_bmpFHeader.bfReserved1=0; //fixed
    m_bmpFHeader.bfReserved2=0; //fixed
    m_bmpFHeader.bfOffBits=sizeof(BITMAPINFOHEADER)+sizeof(BITMAPFILEHEADER)
        +m_numColors*sizeof(RGBQUAD);

    // Allocate pixel array, set display pixels to 0
    try
    {
        m_Pixels=new BYTE[m_numPixelBytes];
    }
    catch(...)
    {
        AfxMessageBox("Image error: out of memory", MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    if(!m_Pixels)
    {
        AfxMessageBox("Image error: out of memory", MB_OK|MB_ICONEXCLAMATION);
        return false;
    }
    else    memset(m_Pixels,0,m_numPixelBytes);

```

```

// Grab main window state bar
CMDIFrameWnd* main_frame=(CMDIFrameWnd*)AfxGetMainWnd();
m_MainStatusBar=(CStatusBar*)(main_frame->GetMessageBar());

// Set logical palette
if(!pPal) // create independent palette
{
    m_pPal=new Palette(theApp.app_Metheus,m_maxPixelValue,m_RGB);
    m_ReleasePalette=true;
}
else // link to existing palette
{
    m_pPal=pPal;
    m_ReleasePalette=false;
}

m_DisplayFailed=false; m_UpdateBitmap=true; m_Bitmap=NULL;

return true;
}

/*****
 *
 * Set image palette (will be set on devices supporting palettes)
 * Must be called for 8-bit bitmaps
 *****/
//Set palette to specified color map array
bool Image::SetPalette(long* color_map, long color_map_size, bool show_progress)
{
    long i, r, g, b, p;

    if(m_pPal->p_active) // if using logical palettes
    {
        return m_pPal->SetPalette(color_map, color_map_size);
    }
    else // directly reset pixel values
    {
        // Display progress control in the main frame status bar
        if(show_progress) theApp.ShowProgress(1,"Modifying pixel values ...");
        // Backup old pixel values
        ResetPixels(true);
        // Remap pixel values
        if(m_RGB)
        {
            for(i=0; i<(long)m_numPixels; i++)
            {
                if(show_progress && i%100==0) theApp.ShowProgress((99*i)/m_numPixels);
                p=GetPixel(i);
                r=GetRValue(p); if(r>=color_map_size) r=color_map_size-1;
                g=GetGValue(p); if(g>=color_map_size) g=color_map_size-1;
                b=GetBValue(p); if(b>=color_map_size) b=color_map_size-1;
                SetPixel(i,RGB(color_map[r],color_map[g],color_map[b]));
            }
        }
        else // greyscale
        {
            for(i=0; i<(long)m_numPixels; i++)
            {
                if(show_progress && i%100==0) theApp.ShowProgress((99*i)/m_numPixels);
                p=GetPixel(i); if(p>=color_map_size) p=color_map_size-1;
                SetPixel(i,color_map[p]);
            }
        }
        return true;
    }
}

bool Image::SetPalette(int px, int py, int pxmax, int pymax) // set palette with respect to the
// mouse position
{

```



```

int offset=((long)m_maxValue/3)*(2*px-pxmax)/pxmax;
double x=(2.0*py)/pymax;
x=1+2*(x-1)*(x-1)*(x-1);
return m_pPal->SetPalette(offset,x);
}

```

```

/*****
*
*   Plot DIB pixels
*
*****/
bool Image::DisplayDIB(CDC *pDC, CRect crectScreenArea, CRect crectImageArea,
                      CPoint pScroll, bool optimize)
{
    if(m_DisplayStatus == DisplayHidden)    return true;    // no display for hidden
    UINT palUsage;

    if(m_DisplayFailed) return false;
    if(m_pPal->p_active && m_pPal->LoadPalette(pDC,m_RGB))
    {
        palUsage=DIB_PAL_COLORS;
    }
    else palUsage=DIB_RGB_COLORS;

    // Printing ?
    if(pDC->IsPrinting())
    {
        optimize=false;
        int list_width  = pDC->GetDeviceCaps(HORZRES);
        int list_height = pDC->GetDeviceCaps(VERTRES);
        double mag = 0.9*min( (double)(list_width/GetWidth()),
                             (double)(list_height/GetHeight()));
        int print_width =(int)(mag*GetWidth());
        int print_height=(int)(mag*GetHeight());
        crectScreenArea=CRect(
            CPoint((list_width-print_width)/2,(list_height-print_height)/2),
            CSize(print_width,print_height));
        crectImageArea =CRect(CPoint(0,0),CSize(GetWidth(),GetHeight()));
        pScroll = CPoint(0,0);
    }

    /* Optimize drawing regions */
    CRect rcClip, rcDraw;
    pDC->GetClipBox(rcClip);
    rcClip.NormalizeRect();
    if(optimize)
    {
        CRect rcDIB;

        if(rcClip.IsRectEmpty())    return true;
        rcClip.InflateRect(4,4);
        rcDraw.IntersectRect(rcClip,crectScreenArea);
        rcDraw.NormalizeRect();
        if(rcDraw.IsRectEmpty())    return true;

        rcDIB=m_ScreenMap.Screen_to_Image(rcDraw);
        if(rcDIB.IsRectEmpty())    return true;

        crectScreenArea.CopyRect(rcDraw);
        if(theApp.app_Metheus) crectScreenArea.OffsetRect(-pScroll);
        crectImageArea.CopyRect(rcDIB);
    }

    /* Correct image area */
    if(crectImageArea.left<0) crectImageArea.OffsetRect(-crectImageArea.left,0);
    else if (crectImageArea.right>m_Width)
        crectImageArea.OffsetRect(m_Width-crectImageArea.right,0);
    if(crectImageArea.top <0) crectImageArea.OffsetRect(0,-crectImageArea.top);
    else if (crectImageArea.bottom>m_Height)
        crectImageArea.OffsetRect(0,m_Height-crectImageArea.bottom);
}

```

```

/* Set screen (destination) and bitmap (source) areas */
long xDest=crectScreenArea.TopLeft().x; long yDest=crectScreenArea.TopLeft().y;
long wDest=crectScreenArea.Width();      long hDest=crectScreenArea.Height();
long xBmp=crectImageArea.TopLeft().x;
long yBmp;
if(theApp.app_Metheus) yBmp=crectImageArea.TopLeft().y;
else
{
    yBmp=m_Height-crectImageArea.TopLeft().y-crectImageArea.Height();
}
long wBmp=crectImageArea.Width();      long hBmp=crectImageArea.Height();

/* Display as DIB */
if(wDest==0 || hDest==0 || wBmp ==0 || hBmp==0) return true;
if(!theApp.app_Metheus)
{
    m_DisplayFailed= (StretchDIBits( pDC->GetSafeHdc(),
                                   xDest, yDest, wDest, hDest, xBmp, yBmp, wBmp, hBmp,
                                   m_Pixels, m_bmpInfo, palUsage, SRCCOPY)<=0);
}
else // Metheus
{
    m_DisplayFailed=true;
    if(m_UpdateBitmap)
    {
        if(m_Bitmap)
            MetheusDeleteCompatibleBitmap(pDC->GetSafeHdc(),m_Bitmap);
        m_Bitmap = MetheusCreateCompatibleBitmap(pDC->GetSafeHdc(),GetWidth(), GetHeight());
        if(MetheusLoadImageFromData(pDC->GetSafeHdc(), m_Bitmap,
            theApp.app_DynamicPaletteStart,m_Pixels, GetWidth(), GetHeight(),
            GetWidth()*m_Bytes_per_Pixel,8*m_Bytes_per_Pixel,
            0, 0, GetWidth(), GetHeight(), 0,0)==FALSE)
        {
            AfxMessageBox("Cannot create image bitmap", MB_OK | MB_ICONEXCLAMATION);
            return false;
        }
    }
    m_DisplayFailed=
        (MetheusStretchBltImage(pDC->GetSafeHdc(), NULL,
                               xDest, yDest, wDest, hDest,
                               m_Bitmap,
                               xBmp, yBmp,wBmp,hBmp,SRCCOPY)==FALSE);
}

/* Display selection frame */
if(m_DisplayStatus == DisplaySelected)
{
    CRect r2 = m_ScreenMap.crScreen;
    pDC->DrawEdge(r2,EDGE_BUMP, BF_RECT);
}

/* Display image info, if needed */
if(m_ShowImageInfo) DisplayImageInfo(pDC);

/* Any display errors ?? */
if(m_DisplayFailed)
{
    DWORD ecode=GetLastError();
    CString estr;
    estr.Format(" GDI error code: %ld\n Please reload the image", ecode);
    AfxMessageBox(estr, MB_OK | MB_ICONEXCLAMATION);
}
m_UpdateBitmap=false;
return (!m_DisplayFailed);
}

bool Image::DisplayDIB(CDC *pDC)
{
    return DisplayDIB(pDC,m_ScreenMap.crScreen,m_ScreenMap.crImage, CPoint(0,0), true);
}

bool Image::DisplayDIB(CDC *pDC, CPoint pScroll)
{
    return DisplayDIB(pDC,m_ScreenMap.crScreen,m_ScreenMap.crImage, pScroll, true);
}

```

```

/*****
*
*   Display image caption over the image
*
*****/
void Image::DisplayImageInfo(CDC *pDC)
{
    CString info;
    FormatImageInfo(info);
    if(pDC->IsPrinting())
    {
        pDC->SetMapMode(MM_LOENGLISH);
        CFont *oldcf = pDC->SelectObject(&theApp.app_MediumFont);
        pDC->SetTextColor(RGB(0,0,0));
        pDC->TextOut(0,0,info);
        pDC->SelectObject(oldcf);
        pDC->SetMapMode(MM_TEXT);
    }
    else
    {
        pDC->SetMapMode(MM_TEXT);
        CFont *oldcf = pDC->SelectObject(&theApp.app_SmallFont);
        // Find caption rectangle sizes
        CRect r=CRect(0,0,10,10);
        r.OffsetRect(m_ScreenMap.crScreen.TopLeft()+CSize(5,5));
        pDC->DrawText(info,r,DT_CALCRECT);
        r &= m_ScreenMap.crScreen;
        // Find out the best caption color
        COLORREF c1 = pDC->GetPixel(r.TopLeft());
        COLORREF c2 = pDC->GetPixel(r.BottomRight());
        COLORREF c3 = pDC->GetPixel(r.CenterPoint());
        COLORREF c = RGB(
            (GetRValue(c1)+GetRValue(c2)+GetRValue(c3))/3,
            (GetGValue(c1)+GetGValue(c2)+GetGValue(c3))/3,
            (GetBValue(c1)+GetBValue(c2)+GetBValue(c3))/3);
        BYTE g = (GetRValue(c)+GetGValue(c)+GetBValue(c))/3;
        if(g<100) g = 255; else g = 0;
        pDC->SetTextColor(RGB(0,g,0));
        pDC->SetBkMode(TRANSPARENT);
        // Display the caption
        pDC->DrawText(info,r,DT_LEFT | DT_END_ELLIPSIS);
        pDC->SelectObject(oldcf);
    }
}

/*****
*
*   Map pixel coordinates from 2D to linear array of pixel bytes
*
*****/
inline unsigned long Image::MapPixel (unsigned int x, unsigned int y)
{
    /* Safe pixel mapping - important ! */
    if(x >= (unsigned int)m_Width) x=m_Width-1;
    if(y >= (unsigned int)m_Height) y=m_Height-1;

    /* Metheus vertical flip */
    if(theApp.app_Metheus) y=m_Height-1-y;

    return m_Bytes_per_Pixel*(m_Width*(long)(m_Height-y-1)+x);
}

/*****
*
*   Converts DIB to DDB
*
*****/
HBITMAP Image::DIB_to_DDB(CDC *pDC)
{
    return CreateDIBitmap(pDC->GetSafeHdc(),&(m_bmpInfo->bmiHeader),
        CBM_INIT,m_Pixels,m_bmpInfo,DIB_RGB_COLORS);
}

```

```

/*****
 *
 * Returns subimage of the current image
 *
 *****/
void Image::GetSubimage(Image* sub, const int xleft, const int ytop)
{
    if(!sub) return;
    // Backup pixel values
    sub->ResetPixels(true);

    int x,y;
    for(x=xleft; x<xleft+(sub->GetWidth()); x++)
    {
        for(y=ytop; y<ytop+(sub->GetHeight()); y++)
        {
            if(x<0 || x>=this->GetWidth() || y<0 || y>=this->GetHeight())
            {
                sub->SetPixel(x-xleft,y-ytop,0); //black background color
            }
            else
            {
                sub->SetPixel(x-xleft,y-ytop,this->GetPixel(x,y));
            }
        }
    }
}

/*****
 *
 * Set image flip parameter
 *
 *****/
void Image::TR_Flip(bool horizontal)
{
    int x, y, w=GetWidth(), h=GetHeight();
    long p;
    if(horizontal) // horizontal flip
    {
        ResetPixels(true); // backup
        for(x=0; x<w/2; x++)
        {
            ::ShowProgress((200*x)/w, "Flipping horizonatlly ..");
            for(y=0; y<h; y++)
            {
                p = GetPixel(x,y);
                SetPixel(x,y,GetPixel(w-x,y));
                SetPixel(w-x,y,p);
            }
        }
    }
    else
    {
        ResetPixels(true); // backup
        for(y=0; y<h/2; y++)
        {
            ::ShowProgress((200*y)/h, "Flipping vertically ..");
            for(x=0; x<w; x++)
            {
                p = GetPixel(x,y);
                SetPixel(x,y,GetPixel(x,h-y));
                SetPixel(x,h-y,p);
            }
        }
    }
    ::ShowProgress(0, "Ready");
    Beep(500,100);
}

```

```

/*****
 *
 *   Increase image brightness by b percent
 *   (decreases brightness for negative b)
 *
 *****/
void Image::SetBrightness(int b)
{
    long amin, amax, bmin, bmax;

    /* Validation */
    if(b<-99) b=-99; else if(b>99) b=99;
    if(b==0) return; // nothing to do;

    /* Find current max and min */
    if(m_pPal->p_active) m_pPal->Get_Pal_minmax(amin,amax);
    else Get_Pixel_minmax(amin,amax);

    /* Set new min and max */
    bmin=(int)(amin+(((long)b)*(amax-amin))/100);
    bmax=bmin+(amax-amin);
    if(bmin<0) bmin=0;
    if(bmax>m_maxPixelValue) bmax=m_maxPixelValue;

    /* Remap the pixels */
    TR_HistStretch(amin,amax,bmin,bmax);
    return;
}

/*****
 *
 *   Apply 3x3 Smooth mask:
 *
 *   1  2  1
 *   2  4  2
 *   1  2  1
 *
 *****/
long Image::TR_Smooth(const int x, const int y)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);    long a20=GetPixel(x+1,y-1);
    long a01=GetPixel(x-1,y);      long a11=GetPixel(x,y);      long a21=GetPixel(x+1,y);
    long a02=GetPixel(x-1,y+1);    long a12=GetPixel(x,y+1);    long a22=GetPixel(x+1,y+1);
    if(!m_RGB)
    {
        long t=(a11<<2)+((a01+a12+a21+a10)<<1)+a02+a22+a00+a20;
        return (t>>4);
    }
    else
    {
        long tb,tg,tr;
        tb=(GetBValue(a11)<<2)
            +((GetBValue(a01)+GetBValue(a12)+GetBValue(a21)+GetBValue(a10))<<1)
            +GetBValue(a02)+GetBValue(a22)+GetBValue(a00)+GetBValue(a20);
        tg=(GetGValue(a11)<<2)
            +((GetGValue(a01)+GetGValue(a12)+GetGValue(a21)+GetGValue(a10))<<1)
            +GetGValue(a02)+GetGValue(a22)+GetGValue(a00)+GetGValue(a20);
        tr=(GetRValue(a11)<<2)
            +((GetRValue(a01)+GetRValue(a12)+GetRValue(a21)+GetRValue(a10))<<1)
            +GetRValue(a02)+GetRValue(a22)+GetRValue(a00)+GetRValue(a20);
        return RGB(min(tr>>4,255),min(tg>>4,255),min(tb>>4,255));
    }
}

/*****
 *
 *   Apply SUSAN-like or 3x3 Gaussian mask to denoise:
 *
 *   1  4  1
 *   4 12  4
 *
 *****/

```

```

*   1   4   1
*
*****/
long Image::TR_DeNoise(const int x, const int y)
{
    static const int    rad=2;
    static const int    rad2=rad*2+1;

    static bool        init_state=false;
    static int          thresh=rad;
    static long         bp[50];
    static long         dpt[rad2][rad2];
    static double       thresh2=1.0/(thresh*thresh);

    long    sigma=50, brightness, ltmp, area, total;

    // Calculate exponent and distance look-up tables
    if (!init_state)
    {
        for(int k=0;k<50;k++)    bp[k]=(long)(100.0*exp(-0.1*k));
        for(int i=-rad; i<=rad; i++)
        {
            for(int j=-rad; j<=rad; j++)
            {
                dpt[i+rad][j+rad] = (long) (100.0 * exp((-i*i-j*j)*thresh2));
            }
        }
        init_state=true;
    }

    // Calculate image squared variance "sigma"
    if((x==rad) && (y==rad))
    {
        GetStatistics(area, sigma, ltmp);
        if(sigma<1) sigma=1;
    }

    // Get central pixel
    long all=GetPixel(x,y);

    // Do denoising
    if(!m_RGB) // Process grayscale image
    {
        area = 0;
        total = 0;
        for(int k=0; k<rad2; k++)
        {
            for(int l=0; l<rad2; l++)
            {
                brightness=GetPixel(x+k-rad,y+l-rad);
                ltmp=brightness-all;
                ltmp=(10*ltmp*ltmp)/sigma;
                if (ltmp>=50)    continue;
                ltmp=bp[ltmp];
                ltmp *= dpt[k][l];
                area += ltmp;
                total += ltmp * brightness;
            }
        }
        ltmp = area-10000;

        if (ltmp==0)
        {
            return ((all<<2)+GetPixel(x-1,y)+GetPixel(x+1,y)+
                    GetPixel(x,y-1)+GetPixel(x,y+1))>>3;
        }
        else
            return (total-(all*10000))/ltmp;
    }
    else// Process color image
    {
        long resultr, resultg, resultb,br1;
        long arear,areag,areab;
        long totalr,totalg,totalb;
    }
}

```

```

arear = areag = areab = 0;
totalr = totalg = totalb = 0;

for(int k=0; k<rad2; k++)
{
    for(int l=0; l<rad2; l++)
    {
        brightness=GetPixel(x+k-rad,y+l-rad);
        //-----red-----
        br1=GetRValue(brightness);
        ltmp=br1-GetRValue(a11);
        ltmp=(10*ltmp*ltmp)/sigma;
        if (ltmp<50)
        {
            ltmp=bp[ltmp];
            ltmp *= dpt[k][l];
            arear += ltmp;
            totalr += ltmp * br1;
        }
        //-----green-----
        br1=GetGValue(brightness);
        ltmp=br1-GetGValue(a11);
        ltmp=(10*ltmp*ltmp)/sigma;
        if (ltmp<50)
        {
            ltmp=bp[ltmp];
            ltmp *= dpt[k][l];
            areag += ltmp;
            totalg += ltmp * br1;
        }
        //-----blue-----
        br1=GetBValue(brightness);
        ltmp=br1-GetBValue(a11);
        ltmp=(10*ltmp*ltmp)/sigma;
        if (ltmp<50)
        {
            ltmp=bp[ltmp];
            ltmp *= dpt[k][l];
            areab += ltmp;
            totalb += ltmp * br1;
        }
    }
}

//-----red-----
ltmp = arear-10000;
if (ltmp==0)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);
    long a20=GetPixel(x+1,y-1);    long a01=GetPixel(x-1,y);

    long a21=GetPixel(x+1,y);      long a02=GetPixel(x-1,y+1);
    long a12=GetPixel(x,y+1);      long a22=GetPixel(x+1,y+1);
    resultr=(GetRValue(a11)<<3)
        +(GetRValue(a11)+GetRValue(a01)+GetRValue(a12)+GetRValue(a21)+
        GetRValue(a10))<<2)
        +GetRValue(a02)+GetRValue(a22)+GetRValue(a00)+GetRValue(a20);
    if (resultr<0) resultr=0;
    else resultr >= 5;
}
else    resultr = (totalr-(GetRValue(a11)*10000))/ltmp;

//-----green-----
ltmp = areag-10000;
if (ltmp==0)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);
    long a20=GetPixel(x+1,y-1);    long a01=GetPixel(x-1,y);

    long a21=GetPixel(x+1,y);      long a02=GetPixel(x-1,y+1);
    long a12=GetPixel(x,y+1);      long a22=GetPixel(x+1,y+1);
    resultg=(GetGValue(a11)<<3)
        +(GetGValue(a11)+GetGValue(a01)+GetGValue(a12)+GetGValue(a21)+
        GetGValue(a10))<<2)
        +GetGValue(a02)+GetGValue(a22)+GetGValue(a00)+GetGValue(a20);
    if (resultg<0) resultg=0;
}

```

```

        else resultg >= 0;
    }
    else    resultg = (totalg - (GetGValue(a11)*10000))/ltmp;
    //-----blue-----
    ltmp = areab-10000;
    if (ltmp==0)
    {
        long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);
        long a20=GetPixel(x+1,y-1);    long a01=GetPixel(x-1,y);

        long a21=GetPixel(x+1,y);    long a02=GetPixel(x-1,y+1);
        long a12=GetPixel(x,y+1);    long a22=GetPixel(x+1,y+1);
        resultb=(GetBValue(a11)<<3)
            +((GetBValue(a11)+GetBValue(a01)+GetBValue(a12)+GetBValue(a21)+
              GetBValue(a10))<<2)
            +GetBValue(a02)+GetBValue(a22)+GetBValue(a00)+GetBValue(a20);
        if (resultb<0) resultb=0;
        else resultb >= 5;
    }
    else    resultb = (totalb - (GetBValue(a11)*10000))/ltmp;
    //-----total color -----
    //return RGB(min(resultr,255),min(resultg,255),min(resultb,255));
    return resultr;
}

}

/*****
 *
 * Apply 3x3 sharpening mask:
 *
 * -1  -1  -1
 * -1  10  -1
 * -1  -1  -1      /      2
 *
 *****/
Long Image::TR_Sharp(const int x, const int y)
{
    long a00=GetPixel(x-1,y-1);    long a10=GetPixel(x,y-1);    long a20=GetPixel(x+1,y-1);
    long a01=GetPixel(x-1,y);    long a11=GetPixel(x,y);    long a21=GetPixel(x+1,y);
    long a02=GetPixel(x-1,y+1);    long a12=GetPixel(x,y+1);    long a22=GetPixel(x+1,y+1);
    if(!m_RGB)
    {
        long t=((a11<<1)+(a11<<3) - (a01+a12+a21+a10+a02+a22+a00+a20))>>1;
        if(t<0) t=0;
        else if(t>m_maxPixelValue) t=m_maxPixelValue;
        return t;
    }
    else
    {
        long tr,tg,tb;
        tr=( (GetRValue(a11)<<1)+(GetRValue(a11)<<3)
            - (GetRValue(a01)+GetRValue(a12)+GetRValue(a21)+GetRValue(a10)
              +GetRValue(a02)+GetRValue(a22)+GetRValue(a00)+GetRValue(a20)) )>>1;
        if(tr<0) tr=0;
        tg=( (GetGValue(a11)<<1)+(GetGValue(a11)<<3)
            - (GetGValue(a01)+GetGValue(a12)+GetGValue(a21)+GetGValue(a10)
              +GetGValue(a02)+GetGValue(a22)+GetGValue(a00)+GetGValue(a20)) )>>1;
        if(tg<0) tg=0;
        tb=( (GetBValue(a11)<<1)+(GetBValue(a11)<<3)
            - (GetBValue(a01)+GetBValue(a12)+GetBValue(a21)+GetBValue(a10)
              +GetBValue(a02)+GetBValue(a22)+GetBValue(a00)+GetBValue(a20)) )>>1;
        if(tb<0) tb=0;
        return RGB(min(tr,255),min(tg,255),min(tb,255));
    }
}

/*****
 *
 * Apply 3x3 edge detector
 *
 *****/

```



```

// DCM.h : main header file for the DCM application
//

#ifndef AFX_DCM_H__INCLUDED_
#define AFX_DCM_H__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

// #include <afxdao.h>
#include "resource.h" // main symbols
#include "FileBrowser.h" // Added by ClassView
#include "QUERY\QueryRetrieve.h" // Added by ClassView
#include "QUERY\ODBC.h" // Added by ClassView
// #include "LogFile.h" // Added by ClassView

///// User messages
#define WM_USER_DISPLAY_MOST_RECENT WM_USER+31

////////////////////////////////////
// CDCMApp:
// See DCM.cpp for the implementation of this class
//

class CDCMApp : public CWinApp
{
public:
    bool app_Metheus;
    UINT app_ResolutionScaleFactor;
    int app_SupportedPaletteSize;
    ULONG app_DynamicPaletteStart;
    CString app_DirectoryRoot, app_DirectoryTmp,
    app_DirectoryData;
    CSize app_ScreenResolution;
    CPen app_Pen;
    CFont app_SmallFont, app_MediumFont,
    app_LargeFont;
    RTC app_RTC;
    LogFile app_ClientLog, app_ServerLog;
    // DICOMDatabase app_DataBase;
    ODBCDatabase app_DataBase;
    FileBrowser app_FileBrowser;
    QueryRetrieve app_QueryRetrieve;

    static void DisplayRecords(Array<DICOMRecord> &a, bool from_local);
    void ShowProgress(int percent, char* info=NULL);
    void MoveWindowToCorner(CWnd* wnd, CSize offset=CSize(0,0));
    BOOL TestFunction();
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName)
    { return this->OpenDocumentFile(lpszFileName, true); }
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName, bool AddToDatabase);

    CDCMApp();
    ~CDCMApp();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CDCMApp)
public:
    virtual BOOL InitInstance();
//}}AFX_VIRTUAL

// Implementation

//{{AFX_MSG(CDCMApp)
afx_msg void OnAppAbout();
afx_msg void OnFileBrowse();

```

```

afx_msg void OnFileQueryRetrieve();
afx_msg void OnUpdateFileQueryRetrieve(CCmdUI* pCmdUI);
afx_msg void OnDataBaseImportFiles();
afx_msg void OnDataBaseAttachFiles();
afx_msg void OnDataBaseRemoveFilesImported();
afx_msg void OnDataBaseAddRecords();
afx_msg void OnDataBaseRemoveRecords();
afx_msg void OnDataBaseRemoveFilesAttached();
afx_msg void OnDataBaseRemoveALLRecords();
//}}AFX_MSG
afx_msg LRESULT OnDisplayMostRecent(WPARAM wParam, LPARAM lParam);
DECLARE_MESSAGE_MAP()

private:
void SerializeApp(bool is_loading);
OProgress app_Progress;

void LoadStdProfileSettings(UINT nMaxMRU = _AFX_MRU_COUNT);
bool TimeBomb();
bool SetDirectories();
};

extern CDCMApp theApp;
extern void ShowProgress(int percent, char* info);
extern CString Trim(char* s);

////////////////////////////////////
// MemoryLeakTest
// Simple Class to test for memory leaks

class MemoryLeakTest
{
#ifdef _DEBUG
private:
    CMemoryState m_oldMemState, m_newMemState, m_diffMemState;
#endif
public:
    inline void Start()
    {
#ifdef _DEBUG
        m_oldMemState.Checkpoint();
#endif
        return;
    };
    inline void End(int code=0)
    {
#ifdef _DEBUG
        m_newMemState.Checkpoint();
        if( m_diffMemState.Difference( m_oldMemState, m_newMemState ) )
        {
            TRACE( "Memory leaked!\n" );
            m_diffMemState.DumpStatistics();
            CString message;
            message.Format("\nLocation Code = %d", code);
            if(code) AfxMessageBox( "Memory leaked!" + message );
            Beep(900,200); Beep(100,200);
        }
        else
        {
            CString message;
            message.Format("\nLocation Code = %d", code);
            if(code) AfxMessageBox( "No memory leaks" + message );
            Beep(100,100); Beep(100,100);
        }
#endif
        return;
    };
};

////////////////////////////////////
// ORecentFileList document

class ORecentFileList : public CRecentFileList
{

```

1. *What is the purpose of the study?*  
 2. *What are the research questions or hypotheses?*  
 3. *What is the study design?*  
 4. *What are the participants and settings?*  
 5. *What are the variables and measurements?*  
 6. *What are the results and conclusions?*  
 7. *What are the strengths and limitations?*  
 8. *What are the implications for practice?*  
 9. *What are the ethical considerations?*  
 10. *What are the key words?*

```
// DCM.cpp : Defines the class behaviors for the application.
//
```

```
#include "stdafx.h"
#include "DCM.h"
```

```
#include "MainFrm.h"
#include "ChildFrm.h"
#include "DCMDoc.h"
#include "DCMView.h"
#include <direct.h>
#include "CustomFileDialog.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
void ShowProgress(int percent, char* info)
{
    theApp.ShowProgress(percent, info);
};
```

```
CString Trim(char* s)
{
    CString str(s);
    str.TrimLeft(); str.TrimRight();
    return str;
}
```

```
////////////////////////////////////
CDCMApp
```

```
BEGIN_MESSAGE_MAP(CDCMApp, CWinApp)
    //{{AFX_MSG_MAP(CDCMApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    ON_COMMAND(ID_FILE_BROWSE, OnFileBrowse)
    ON_COMMAND(ID_FILE_QUERYRETRIEVE, OnFileQueryRetrieve)
    ON_UPDATE_COMMAND_UI(ID_FILE_QUERYRETRIEVE, OnUpdateFileQueryRetrieve)
    ON_COMMAND(ID_DATABASE_ADDFILES_IMPORT, OnDataBaseImportFiles)
    ON_COMMAND(ID_DATABASE_ADDFILES_ATTACH, OnDataBaseAttachFiles)
    ON_COMMAND(ID_DATABASE_REMOVEFILES, OnDataBaseRemoveFilesImported)
    ON_COMMAND(ID_DATABASE_ADDRECORDS, OnDataBaseAddRecords)
    ON_COMMAND(ID_DATABASE_REMOVEVERECORDS, OnDataBaseRemoveRecords)
    ON_COMMAND(ID_DATABASE_REMOVEFILES_ATTACHED, OnDataBaseRemoveFilesAttached)
    ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
    ON_COMMAND(ID_DATABASE_REMOVEALLRECORDS, OnDataBaseRemoveALLRecords)
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ///no new files/// ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
    ON_THREAD_MESSAGE(WM_USER_DISPLAY_MOST_RECENT, OnDisplayMostRecent)
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CDCMApp construction
```

```
CDCMApp::CDCMApp()
{
    // TODO: add construction code here
    // Place all significant initialization in InitInstance
    app_DirectoryTmp="";
}
CDCMApp::~CDCMApp()
{
    // Delete graphical objects
    app_Pen.DeleteObject();
    app_SmallFont.DeleteObject();
}
```

```

app_MediumFont.DeleteObject();
app_LargeFont.DeleteObject();

// Clean tmp directory
if( app_DirectoryTmp!="") // Tmp directory exists
{
    WIN32_FIND_DATA wf;
    CString nf=app_DirectoryTmp+"\\*";
    HANDLE hf=FindFirstFile(nf,&wf);
    do
    {
        CString tf=CString(wf.cFileName);
        if(tf!="." && tf!="..")
        {
            DeleteFile(app_DirectoryTmp+"/"+tf);
        }
    } while (FindNextFile(hf,&wf));
    FindClose(hf);
}

// Serialize application data
SerializeApp(false);
}
/*****
*
*   Serialize application data
*
*****/
void CDCMApp::SerializeApp(bool is_loading)
{
    // Open data file
    CString fname = "\\app.dat";
    CString f = app_DirectoryData+fname;
    FILE* fp;
    fp = fopen((char*)(LPCSTR)f, is_loading ? "rb" : "wb");
    if(!fp) return;

    // Serialize
    ((ORecentFileList*)m_pRecentFileList)->SerializeORFLList(fp, is_loading);

    // Close data file
    fclose(fp);

    ////////////////////////////////////////
    // The one and only CDCMApp object
    ////////////////////////////////////////

    CDCMApp theApp;
    ////////////////////////////////////////
    // CDCMApp initialization
    BOOL CDCMApp::InitInstance()
    {
        // Ensure that only one DCM copy is running
        HANDLE hMutex = CreateMutex ( NULL, FALSE, "DCM_UNIQUE_MUTEX");
        if ( NULL != hMutex && ERROR_ALREADY_EXISTS == GetLastError() )
        {
            CWnd * hWnd = CWnd::FindWindow( NULL,"DCM_UNIQUE_MUTEX");
            if ( hWnd != NULL )
            {
                // if found make it the current window
                hWnd->SetForegroundWindow();
            }
            AfxMessageBox("DCM is already running !");
            return FALSE;
        }

        // Check the time bomb
        if(!TimeBomb()) return FALSE;

        // Socket support
        if (!AfxSocketInit())

```

```

{
    AfxMessageBox("Window sockets failed");
    return FALSE;
}

AfxEnableControlContainer();

// Standard initialization
#ifdef _AFXDLL
    Enable3dControls();          // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();    // Call this when linking to MFC statically
#endif

// Change the registry key under which our settings are stored.
// You should modify this string to be something appropriate
// such as the name of your company or organization.
SetRegistryKey(_T("DCM Workstation"));
LoadStdProfileSettings(); // Load standard INI file options (including MRU)

// Register the application's document templates. Document templates
// serve as the connection between documents, frame windows and views.

CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_DCMTYPE,
    RUNTIME_CLASS(CDCMDoc),
    RUNTIME_CLASS(CChildFrame), //custom MDI child frame
    RUNTIME_CLASS(CDCMView));
AddDocTemplate(pDocTemplate);

// create main MDI Frame window
CMainFrame* pMainFrame = new CMainFrame; // does not need any "delete" later
if (!pMainFrame->LoadFrame(IDR_MAINFRAME)) return FALSE;
m_pMainWnd = pMainFrame;

// Set application graphics parameters
HDC scrHdc=GetDC(NULL); // grab screen DC GetDC
app_Metheus = (IsMetheusDevice(scrHdc)==TRUE);
app_ScreenResolution.cx=::GetDeviceCaps(scrHdc,HORZRES);
app_ScreenResolution.cy=::GetDeviceCaps(scrHdc,VERTRES);
app_ResolutionScaleFactor=__min(app_ScreenResolution.cx/800,
                                app_ScreenResolution.cx/600);
if(app_ResolutionScaleFactor<1) app_ResolutionScaleFactor=1;
app_Pen.CreatePen(PS_SOLID,2*theApp.app_ResolutionScaleFactor,
                  RGB(250,250,250));
int fac=__min(2,theApp.app_ResolutionScaleFactor);
int pix20mm = (25*app_ScreenResolution.cy)/::GetDeviceCaps(scrHdc,VERTSIZE);
app_SmallFont.CreatePointFont(pix20mm, "Arial", NULL);
app_MediumFont.CreatePointFont((3*pix20mm)/2,"Swiss", NULL);
app_LargeFont.CreatePointFont(2*pix20mm, "Swiss", NULL);
if(app_Metheus)
{
    app_SupportedPaletteSize=4096;
    MetheusEscDEVINFO mdinf;
    MetheusGetDEVINFO(scrHdc,&mdinf);
    app_DynamicPaletteStart=mdinf.StartExtraPal+app_SupportedPaletteSize;
}
else
{
    app_DynamicPaletteStart=0;
    if(GetDeviceCaps(scrHdc,RASTERCAPS)&RC_PALETTE)
    {
        app_SupportedPaletteSize=GetDeviceCaps(scrHdc,SIZEPALETTE);
        if(app_SupportedPaletteSize<255) app_SupportedPaletteSize=0;
    }
    else app_SupportedPaletteSize=0;
}
ReleaseDC(NULL,scrHdc);

// Parse command line for standard shell commands, DDE, file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

```

```

// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

// The main window has been initialized, so show and update it.
pMainFrame->ShowWindow(m_nCmdShow);
pMainFrame->UpdateWindow();
// Do not initialize an empty document
//pMainFrame->MDIGetActive()->MDIDestroy();
pMainFrame->MDIGetActive()->GetActiveDocument()->OnCloseDocument();

// Set working directories
if(!SetDirectories()) return FALSE;

// Test some code
if(!TestFunction()) return FALSE;

// Initialize File Browser
if(!app_FileBrowser.Initialize(app_DirectoryTmp)) return FALSE;

// Initialize progress control
if(!app_Progress.Initialize()) return FALSE;

// Load RTC
CString path=app_DirectoryRoot;
path=app_DirectoryRoot.Left(app_DirectoryRoot.GetLength()-6); // remove "/Applic"
path+=CString("dcmdict.txt");
bool dictEnabled = (app_RTC.AttachRTC((char*)(LPCSTR)path, ::ShowProgress)==TRUE);
((CMainFrame*)m_pMainWnd)->SetDictionaryStatus(dictEnabled);

// Initialize log files (must go after RTC and directory initialization)
if(!app_ClientLog.CreatedVL(
    (char*)(LPCSTR)(theApp.app_DirectoryData+"\\ClientLog.txt"),&app_RTC))
{
    AfxMessageBox("Cannot initialize client log");
    return FALSE;
}
if(!app_ServerLog.CreatedVL(
    (char*)(LPCSTR)(theApp.app_DirectoryData+"\\ServerLog.txt"),&app_RTC))
{
    AfxMessageBox("Cannot initialize server log");
    return FALSE;
}
if(!app_QueryRetrieve.InitializeQueryRetrieve(&app_ClientLog,
    &app_ServerLog, &app_DataBase))
{
    return FALSE;
}

// Load serialized data
SerializeApp(true);

return TRUE;
}
/*****
*
* Set directory structure
*
*****/
bool CDCMApp::SetDirectories()
{
    bool newdir=false;
    app_DirectoryTmp="";
    // Get root directory
    app_DirectoryRoot=GetProfileString("Directories","Root","");
    if(app_DirectoryRoot.Find("Profile")>0) app_DirectoryRoot=""; // avoid desktop
    if(app_DirectoryRoot=="") // reset
    {
        newdir=true;
        app_DirectoryRoot=CString(this->m_pszHelpFilePath);
        app_DirectoryRoot.TrimRight();
        app_DirectoryRoot.Replace("/", "\\");
        int n=app_DirectoryRoot.ReverseFind('\\');
        if(n>0) app_DirectoryRoot=app_DirectoryRoot.Left(n);
    }
}

```

```

    app_DirectoryRoot += "\\Applic";
    WriteProfileString("Directories", "Root", app_DirectoryRoot);
}
// Can we access root directory ?
//if(SetCurrentDirectory(app_DirectoryRoot)==FALSE)
if(!::CreateAndSetCurrentDirectory(
    (char*)(LPCSTR)app_DirectoryRoot))
{
    WriteProfileString("Directories", "Root", "");
    AfxMessageBox("Failed to setup the application directory");
    return false;
}
// Create subdirectories
app_DirectoryRoot.Replace("/", "\\");
app_DirectoryTmp = app_DirectoryRoot + "\\temp";
app_DirectoryData = app_DirectoryRoot + "\\data";
CreateDirectory(app_DirectoryTmp, NULL);
CreateDirectory(app_DirectoryData, NULL);
CString dbdir = app_DirectoryRoot + "\\DataBase";
if(!app_DataBase.InitializeDataBase((char*)(LPCSTR)dbdir, &DisplayRecords))
{
    WriteProfileString("Directories", "Root", "");
    return false;
}
if(newdir)
{
    CString info;
    AfxFormatString1(info, IDS_DIRECTORY_SETUP, app_DirectoryRoot);
    AfxMessageBox(info, MB_ICONINFORMATION | MB_OK);
}
return true;
}

*****
* Open (create) a new document
*****
Document* CDCMApp::OpenDocumentFile(LPCTSTR lpszFileName, bool AddToDatabase)
{
    Beep(700, 200);
    if(!lpszFileName || lpszFileName=="") return NULL;
    ShowProgress(5, "Parcing image file...");
    CDCMDoc* pDoc = (CDCMDoc*)(CWinApp::OpenDocumentFile(lpszFileName));
    if(!pDoc) return NULL;
    if(pDoc->IsEmpty())
    {
        pDoc->OnCloseDocument(); // kill empty or invalid documents
        ShowProgress(0, "Ready");
        return NULL;
    }
    else // Looks like a nice DICOM document
    {
        ((ORecentFileList*)m_pRecentFileList)->AddMenuAlias(lpszFileName,
            pDoc->GetMRUAlias());
        if(AddToDatabase)
        {
            app_DataBase.DBAdd( *(pDoc->GetDICOMRecordPtr()) );
        }
        ShowProgress(0, "Ready");
        return pDoc;
    }
}

LRESULT CDCMApp::OnDisplayMostRecent(WPARAM wParam, LPARAM lParam)
{
    if(!wParam) return 0L;
    Array<DICOMRecord>* a = (Array<DICOMRecord>*)wParam;
    if(!a) return 0L;
    if(a->GetSize()<=0) return 0L;
    CDCMDoc* pDoc = NULL;

    // Any documents open ?
    POSITION pos = theApp.GetFirstDocTemplatePosition();
    CMultiDocTemplate* pDocTemplate = NULL;

```



```

    if(pos)
    {
        pDocTemplate = (CMultiDocTemplate*)
            (theApp.GetNextDocTemplate(pos));
    }

    // Go through all existing documents
    if(pDocTemplate)
    {
        POSITION posDoc = pDocTemplate->GetFirstDocPosition();
        while(posDoc && a->GetSize()>0)
        {
            pDoc = (CDCMDoc*) (pDocTemplate->GetNextDoc(posDoc));
            if(!pDoc) break;
            pDoc->AddDICOMRecords(*a);
            pDoc->OnViewRefresh();
        }
    }

    // Open new documents, if needed
    while(a->GetSize()>0)
    {
        pDoc = (CDCMDoc*) (
            theApp.OpenDocumentFile(a->Get(0).GetFileName(), false));
        a->RemoveAt(0);
        if(!pDoc) continue;
        pDoc->AddDICOMRecords(*a);
        pDoc->OnViewRefresh();
    }
    return 0L;
}

void CDCMApp::DisplayRecords(Array<DICOMRecord> &a, bool from_local)
{
    theApp.PostThreadMessage(WM_USER_DISPLAY_MOST_RECENT, (WPARAM)&a,
        (LPARAM)from_local);

    while(a.GetSize()>0) Sleep(100);

    *****
    Launch DICOM browser
    *****/
void CDCMApp::OnFileBrowse()
{
    if(app_FileBrowser.DoModal()==IDOK)
    {
        this->OpenDocumentFile(app_FileBrowser.m_RequestedFile, true);
    }
}

/*****
*
* Start or Terminate Query/Retrieve session
*
*****/
void CDCMApp::OnFileQueryRetrieve()
{
    app_QueryRetrieve.SwitchAppearance();
}

void CDCMApp::OnUpdateFileQueryRetrieve(CCmdUI* pCmdUI)
{
    static bool state=false;
    bool check = (app_QueryRetrieve.GetSafeHwnd() != NULL &&
        app_QueryRetrieve.IsIconic() == FALSE);
    pCmdUI->SetCheck(check);
    if(check != state)
    {
        state=check;
        CMainFrame* mf = (CMainFrame*)AfxGetMainWnd();
        if(mf) mf->EnableLogoAnimation(!check);
    }
}

```

```

}
}
/*****
*
*   DataBase menu handlers
*
*****/
void CDCMApp::OnDataBaseImportFiles()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Files/Directories to Import into the Database");
    if (fd.DoModal() != IDOK )        return;
    int count = fd.GetSelectedCount();
    if(count<=0)        return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);
    for(UINT ind=0; ind < (UINT)count; ind++)
    {
        ShowProgress(90*(ind+1)/count, "Importing ... ");
        app_DataBase.ImportDirectory((char*)(LPCSTR) (fd.GetSelectedAt(ind)),
                                     subdir);
    }
    ShowProgress(0);
    return;
}

void CDCMApp::OnDataBaseAttachFiles()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Files/Directories to Attach to the Database");
    if (fd.DoModal() != IDOK )        return;
    int count = fd.GetSelectedCount();
    if(count<=0)        return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);
    for(UINT ind=0; ind < (UINT)count; ind++)
    {
        ShowProgress(90*(ind+1)/count, "Attaching ... ");
        app_DataBase.AttachDirectory((char*)(LPCSTR) (fd.GetSelectedAt(ind)),
                                     subdir);
    }
    ShowProgress(0);
    return;
}

void CDCMApp::OnDataBaseRemoveFilesImported()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Imported Files/Directories to Remove from the Database");
    if (fd.DoModal() != IDOK )        return;
    int count = fd.GetSelectedCount();
    if(count<=0)        return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);
    for(UINT ind=0; ind < (UINT)count; ind++)
    {
        ShowProgress(90*(ind+1)/count, "Removing selected ... ");
        app_DataBase.UnImportDirectory(
            (char*)(LPCSTR) (fd.GetSelectedAt(ind)) , subdir);
    }
    ShowProgress(0);
    return;
}

void CDCMApp::OnDataBaseRemoveFilesAttached()
{
    // Display modal FileOpen dialog
    CCustomFileDialog fd;
    fd.SetTitle("Select Attached Files/Directories to Remove from the Database");
    if (fd.DoModal() != IDOK )        return;
    int count = fd.GetSelectedCount();
    if(count<=0)        return;
    // Process selected strings
    bool subdir = (fd.m_SelectSubdirectories == TRUE);

```

```

for(UINT ind=0; ind < (UINT)count; ind++)
{
    ShowProgress(90*(ind+1)/count, "Removing attached ... ");
    app_DataBase.UnAttachDirectory((char*)(LPCSTR)(fd.GetSelectedAt(ind)),
        subdir);
}
ShowProgress(0);
return;
}

void CDCMApp::OnDataBaseAddRecords()
{
    app_QueryRetrieve.DoModeless();
}

void CDCMApp::OnDataBaseRemoveRecords()
{
    DQRSearch ds;
    if(ds.DoModal() != IDOK) return;
    if(AfxMessageBox("Delete specified items from local database ?",
        MB_YESNO) != IDYES) return;
    DICOMRecord dr;
    ds.WriteIntoDICOMRecord(dr);
    int n = app_DataBase.DBRemove(dr);
    if(n>0)
    {
        CString info;
        info.Format("%d records were removed from the local database\n\n"
            "Some of the records in the query results window\n"
            "may no longer be valid !",n);
        AfxMessageBox(info, MB_ICONINFORMATION);
    }
    else AfxMessageBox("No matching records found", MB_ICONINFORMATION);
}

void CDCMApp::OnDataBaseRemoveALLRecords()
{
    app_DataBase.RemoveAllRecords();
}

//
// *****
// Move window to right bottom corner of the screen
// (with specified offset)
// Use to display utility dialogs
// *****
//
void CDCMApp::MoveWindowToCorner(CWnd *wnd,
    CSize offset/*=CSize(0,0)*/)
{
    CRect wr;
    wnd->GetWindowRect(wr);
    wr.OffsetRect(app_ScreenResolution-wr.BottomRight()
        -CSize(40,40)-offset);
    wnd->MoveWindow(wr,TRUE);
    wnd->BringWindowToTop();
}

// *****
// *
// * Time bomb
// *
// *****
//
bool CDCMApp::TimeBomb()
{
    CTime t = CTime::GetCurrentTime();
    CTime tstop(2000,12,1,1,1,1);
    if(t>=tstop)
    {
        AfxMessageBox("Program expired !", MB_ICONEXCLAMATION | MB_OK);
        return false;
    }
    return true;
}

```

```

/*****
 *
 *   This function is used for code testing only
 *
 *****/
#include "AccurateTimer.h"
BOOL CDCMApp::TestFunction()
{
    /* MemoryLeakTest mlt;
    mlt.Start();
    {
        DICOMObject d1, d2;
        d1.LoadFromFile("D:\\DICOM\\Data\\0.dcm");
        d2.LoadFromFile("D:\\DICOM\\Data\\00.dcm");

        ImageSeries s1, s2;
        s1.ReadDO(d1);
        s2.ReadDO(d2);
        s1.Append(s2);
    }
    mlt.End(1);
    return FALSE;
*/

    return TRUE;
}
/*****
 *
 *   Displaying progress in the min frame status bar
 *
 *****/
void CDCMApp::ShowProgress(int percent, char* info /* =NULL */)
{
    app_Progress.ShowProgress(percent, info);
}
/*****
 *
 *   Displaying meaninful recent file list names
 *
 *****/
void CDCMApp::LoadStdProfileSettings(UINT nMaxMRU /* = _AFX_MRU_COUNT */)
{
    const TCHAR _afxFileSection[] = _T("Recent File List");
    const TCHAR _afxFileEntry[] = _T("File%d");
    const TCHAR _afxPreviewSection[] = _T("Settings");
    const TCHAR _afxPreviewEntry[] = _T("PreviewPages");
    ASSERT_VALID(this);
    ASSERT(m_pRecentFileList == NULL);

    if (nMaxMRU != 0)
    {
        // create file MRU since nMaxMRU not zero
        m_pRecentFileList = new ORecentFileList(0, _afxFileSection, _afxFileEntry,
            nMaxMRU);
        m_pRecentFileList->ReadList();
    }
    // 0 by default means not set
    m_nNumPreviewPages = GetProfileInt(_afxPreviewSection, _afxPreviewEntry, 0);
}

////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA

```

```

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
    // No message handlers
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CDCMApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

// ORecentFileList
void ORecentFileList::UpdateMenu(CCmdUI *pCmdUI)
{
    ASSERT(m_arrNames != NULL);

    CMenu* pMenu = pCmdUI->m_pMenu;
    if (m_strOriginal.IsEmpty() && pMenu != NULL)
        pMenu->GetMenuString(pCmdUI->m_nID, m_strOriginal, MF_BYCOMMAND);

    if (m_arrNames[0].IsEmpty())
    {
        // no MRU files
        if (!m_strOriginal.IsEmpty())
            pCmdUI->SetText(m_strOriginal);
        pCmdUI->Enable(FALSE);
        return;
    }

    if (pCmdUI->m_pMenu == NULL)
        return;

    for (int iMRU = 0; iMRU < m_nSize; iMRU++)
        pCmdUI->m_pMenu->DeleteMenu(pCmdUI->m_nID + iMRU, MF_BYCOMMAND);

    TCHAR szCurDir[_MAX_PATH];
    GetCurrentDirectory(_MAX_PATH, szCurDir);
    int nCurDir = lstrlen(szCurDir);
    ASSERT(nCurDir >= 0);
    szCurDir[nCurDir] = '\\';
    szCurDir[++nCurDir] = '\\0';
}

```

```

CString strName;
CString strTemp;
CString strMenu="Ku-ku";
for (iMRU = 0; iMRU < m_nSize; iMRU++)
{
    if (!GetDisplayName(strName, iMRU, szCurDir, nCurDir))
        break;

    // double up any '&' characters so they are not underlined
    LPCTSTR lpszSrc = strName;
    LPTSTR lpszDest = strTemp.GetBuffer(strName.GetLength()*2);
    while (*lpszSrc != 0)
    {
        if (*lpszSrc == '&')
            *lpszDest++ = '&';
        if (_istlead(*lpszSrc))
            *lpszDest++ = *lpszSrc++;
        *lpszDest++ = *lpszSrc++;
    }
    *lpszDest = 0;
    strTemp.ReleaseBuffer();
    // Modification: find menu alias
    if(m_menuNames.Lookup(m_arrNames[iMRU], strMenu) == FALSE)
    {
        strMenu=strTemp;
    }

    // insert mnemonic + the file name
    TCHAR buf[10];
    wprintf(buf, _T("&%d "), (iMRU+1+m_nStart) % 10);
    pCmdUI->m_pMenu->InsertMenu(pCmdUI->m_nIndex++,
        MF_STRING | MF_BYPOSITION, pCmdUI->m_nID++,
        CString(buf) + strMenu);
}

// update end menu count
pCmdUI->m_nIndex--; // point to last menu added
pCmdUI->m_nIndexMax = pCmdUI->m_pMenu->GetMenuItemCount();

pCmdUI->m_bEnableChanged = TRUE; // all the added items are enabled
*****
Insert "mname" as a menu alias to existing "fname" file
*****//
void ORecentFileList::AddMenuAlias(LPCTSTR fname, CString mname)
{
    m_menuNames.SetAt(fname, mname);
    CleanMenuAliases();
}

/*****
*
* Remove old menu aliases
*
*****/
void ORecentFileList::CleanMenuAliases()
{
    bool found;
    int i;
    POSITION pos;
    CString key, val;
    for( pos = m_menuNames.GetStartPosition(); pos != NULL; )
    {
        m_menuNames.GetNextAssoc(pos, key, val);
        found=false;
        for(i=0; i<m_nSize; i++)
        {
            if(m_arrNames[i]==key) { found = true; break; }
        }
        if(!found) m_menuNames.RemoveKey(key);
    }
}

```

```

}

/*****
 *
 *   Serialize MRU list aliases
 *
 *****/
void ORecentFileList::SerializeORFLList(FILE* fp, bool is_loading)
{
    if(!fp) return;
    int    n, k;
    char    key[MAX_PATH+1], val[MAX_PATH+1];
    if(is_loading)
    {
        m_menuNames.RemoveAll();
        ::SerializeInteger(fp,n,true);
        if(n<=0) return;
        for(k=0; k<n; k++)
        {
            ::SerializeString(fp, key, true);
            ::SerializeString(fp, val, true);
            m_menuNames.SetAt(key,val);
        }
    }
    else
    {
        CleanMenuAliases();
        n=m_menuNames.GetCount();
        ::SerializeInteger(fp,n,false);
        POSITION    pos;
        CString    ks, vs;
        for( pos = m_menuNames.GetStartPosition(); pos != NULL; )
        {
            m_menuNames.GetNextAssoc(pos, ks, vs);
            sprintf(key,"%s",ks);
            ::SerializeString(fp, key, false);
            sprintf(val,"%s",vs);
            ::SerializeString(fp, val, false);
        }
    }
}

```

```

// ChildFrm.h : interface of CChildFrame class
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_CHILDFRM_H__039FD0CD_68EA_11D2_9576_00105A21774F__INCLUDED_
#define AFX_CHILDFRM_H__039FD0CD_68EA_11D2_9576_00105A21774F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CChildFrame : public CMDIChildWnd
{
    DECLARE_DYNCREATE(CChildFrame)
public:
    CChildFrame();

    // Attributes
public:

    // Operations
public:

    // Overrides
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(CChildFrame)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void ActivateFrame(int nCmdShow = -1);
    }AFX_VIRTUAL

    // Implementation
public:
    virtual ~CChildFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    //{AFX_MSG(CChildFrame)
    afx_msg void OnMove(int x, int y);
    }AFX_MSG
    DECLARE_MESSAGE_MAP()
}

/////////////////////////////////////////////////////////////////

//{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
// line.

#endif // !defined(AFX_CHILDFRM_H__039FD0CD_68EA_11D2_9576_00105A21774F__INCLUDED_)

```



```
// ChildFrm.cpp : implementation of the CChildFrame class
//
```

```
#include "stdafx.h"
#include "DCM.h"
#include "DCMView.h"
```

```
#include "ChildFrm.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CChildFrame
```

```
IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWnd)
```

```
BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWnd)
    //{{AFX_MSG_MAP(CChildFrame)
    ON_WM_MOVE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
////////////////////////////////////
// CChildFrame construction/destruction
```

```
CChildFrame::CChildFrame()
{
    // TODO: add member initialization code here
}
```

```
CChildFrame::~CChildFrame()
```

```
BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
```

```
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
    return CMDIChildWnd::PreCreateWindow(cs);
}
```

```
////////////////////////////////////
// CChildFrame diagnostics
```

```
#ifdef _DEBUG
void CChildFrame::AssertValid() const
{
    CMDIChildWnd::AssertValid();
}
```

```
void CChildFrame::Dump(CDumpContext& dc) const
{
    CMDIChildWnd::Dump(dc);
}
```

```
#endif //_DEBUG
```

```
////////////////////////////////////
// CChildFrame message handlers
```

```
void CChildFrame::ActivateFrame(int nCmdShow)
{
    CMDIChildWnd::ActivateFrame(SW_SHOWMAXIMIZED);
}
```

```
/*
 *
 * Responds to view window moves
 */
```

```
*****
void CChildFrame::OnMove(int x, int y)
{
    CDCMView *pView = (CDCMView*)GetActiveView();
    if(!pView) return;
    pView->EraseTools();

    /*
    // determine desired size of the view
    CRect temp(0,0,0,0);
    pView->CalcWindowRect(temp, CWnd::adjustBorder);
    CalcWindowRect(temp, CWnd::adjustBorder);
    CWnd* pFrameParent = GetParent();

    CRect rcBound;
    pFrameParent->GetClientRect(rcBound);
    //CWinRect rcBound(pFrameParent, CWinRect::CLIENT);

    CRect rectView(CPoint(0, 0), pView->GetTotalSize());
    rectView.right = __min(rectView.right, rcBound.right - temp.Width());
    rectView.bottom = __min(rectView.bottom, rcBound.bottom - temp.Height());
    pView->CalcWindowRect(rectView, CWnd::adjustOutside);
    CalcWindowRect(rectView, CWnd::adjustBorder);
    rectView -= rectView.TopLeft();
    rectView.right = __min(rectView.right, rcBound.right);
    rectView.bottom = __min(rectView.bottom, rcBound.bottom);
    */
}
```

```

// DCMDoc.h : interface of the CDCMDoc class
//
/////////////////////////////////////////////////////////////////

#if !defined(AFX_DCMDOC_H__039FD0CF_68EA_11D2_9576_00105A21774F__INCLUDED_)
#define AFX_DCMDOC_H__039FD0CF_68EA_11D2_9576_00105A21774F__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "MainFrm.h"
#include "Lupa.h" // Added by ClassView
#include "DICOM_H\\DICOMDocument.h" // Added by ClassView

class CDCMDoc : public CDocument, public DICOMDocument
{
protected: // create from serialization only
    CDCMDoc();
    DECLARE_DYNCREATE(CDCMDoc)

public:
    double          m_imageZoom, m_imageZoomFitScreen;
    Lupa            m_Lupa;
    CMainFrame*     m_MainFrame;

    bool            OnZoom(UINT nID, Image *pBmp);
    Image*          GotoImage(int code);
    virtual         ~CDCMDoc();

private:
    CString         m_SoundFileName;

    Image*          GetDIBImage(int n);
    Image*          GetDIBImage();
    CView*          GetViewPtr();

public:
    void OnViewRefresh(BOOL erase_background=FALSE);
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CDCMDoc)
public:
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
protected:
    //{{AFX_MSG(CDCMDoc)
    afx_msg void OnFileSendMail();
    afx_msg void OnUpdateFileSendMail(CCmdUI* pCmdUI);
    afx_msg void OnUpdateZoomFitScreen(CCmdUI* pCmdUI);
    afx_msg void OnUpdateFrameNumber(CCmdUI* pCmdUI);
    afx_msg void SetFitScreenZoom();
    afx_msg void OnViewDicomInfo();
    afx_msg void OnFileSave();
    afx_msg void OnFileSaveAs();
    afx_msg void OnSoundRecord();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
line.

```

```
#endif // !defined(AFX_DCMDOC_039FD0CF_68EA_11D2_9576_00105A21_7__INCLUDED_)
```

039FD0CF\_68EA\_11D2\_9576\_00105A21\_7

```
// DCMDoc.cpp : implementation of the CDCMDoc class
//
```

```
#include "stdafx.h"
#include "DCM.h"
```

```
#include "DCMDoc.h"
#include "DCMView.h"
#include "SoundDialog.h"
#include "Tools/Email.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CDCMDoc
```

```
IMPLEMENT_DYNCREATE(CDCMDoc, CDocument)
```

```
BEGIN_MESSAGE_MAP(CDCMDoc, CDocument)
```

```
//{{AFX_MSG_MAP(CDCMDoc)
ON_UPDATE_COMMAND_UI(ID_ZOOM_FIT_SCREEN, OnUpdateZoomFitScreen)
ON_COMMAND(ID_ZOOM_FIT_SCREEN, SetFitScreenZoom)
ON_COMMAND(ID_VIEW_DICOMINFO, OnViewDicomInfo)
ON_COMMAND(ID_FILE_SAVE, OnFileSave)
ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
ON_COMMAND(ID_SOUND_RECORD, OnSoundRecord)
ON_UPDATE_COMMAND_UI(ID_FRAME_NUMBER, OnUpdateFrameNumber)
//}}AFX_MSG_MAP
ON_COMMAND(ID_FILE_SEND_MAIL, OnFileSendMail)
ON_UPDATE_COMMAND_UI(ID_FILE_SEND_MAIL, OnUpdateFileSendMail)
END_MESSAGE_MAP()
```

```
*****
```

```
*
* Constructor & Destructor
```

```
*****/
```

```
CDCMDoc::CDCMDoc()
```

```
m_MainFrame=(CMainFrame*)AfxGetMainWnd();
m_imageZoomFitScreen=m_imageZoom=1.0; // initial zoom - normal size;
```

```
/* No sound file */
m_SoundFileName=" ";
```

```
}
CDCMDoc::~CDCMDoc() { ; }
```

```
*****
```

```
*
* CDCMDoc serilization
```

```
*****/
```

```
void CDCMDoc::Serialize(CArchive& ar)
```

```
{
    if (ar.IsStoring())
    {
        CFile* pFile = ar.GetFile();
        if(!pFile)
        {
            AfxMessageBox("Cannot open a file");
            return;
        }
        DICOMDocument::SaveDocument(pFile->GetFilePath(),
                                    DICOMDocument::FormatDICOMModified);
        return;
    }
    else
```

```

{
    CFile* pFile = ar.GetFile();
    if(!pFile)
    {
        AfxMessageBox("Cannot open a file");
        return;
    }
    ar.Flush(); // flush all data into the file
    // Can we load the DICOM ?
    if(!DICOMDocument::LoadFile(pFile->GetFilePath())) return;
    //AddDDOFile("D:\\Dicom\\Data\\0.dcm"); // test
    // Do we have any images ?
    if(GetNumberOfImages()<=0)
    {
        AfxMessageBox("This DICOM object contains no image data",
            MB_OK|MB_ICONINFORMATION);
        return;
    }
    // Update image view
    UpdateAllViews(NULL, 0, NULL );
    return;
}

}

/*****
 *
 *   DEBUG diagnostics
 *****/
#ifdef _DEBUG
void CDCMDoc::AssertValid() const
{
    CDocument::AssertValid();
}
void CDCMDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/*****
 *
 *   Get image #n or current frame
 *****/
Image* CDCMDoc::GetDIBImage(int n)
{
    Image* img = DICOMDocument::GetImage(GetCurrentImageIndex());
    if(!img) return NULL;
    if(img)
    {
        m_imageZoom = img->GetZoom();
    }
    img = DICOMDocument::GetImage(n);
    if(img)
    {
        m_MainFrame->ShowFrameNumber(GetCurrentImageIndex()+1);
        img->m_ScreenMap.ValidateZoom(m_imageZoom); // zoom appropriately
    }
    GetCurrentSeries()->SetSelectedImage();
    return img;
}

Image* CDCMDoc::GetDIBImage()
{
    return GetDIBImage(DICOMDocument::GetCurrentImageIndex());
}

/*****
 *
 *   Go to first (code=-2), previous (code=-1),
 *   next (code=+1) or last (code=+2) image
 *****/

```

```

*
*****
Image* CDCMDoc::GotoImage(int code)
{
    switch(code)
    {
        case -2:    return GetDIBImage(0);
        case -1:    return GetDIBImage(GetCurrentImageIndex()-1);
        case 1:     return GetDIBImage(GetCurrentImageIndex()+1);
        case 2:     return GetDIBImage(GetNumberOfImages()-1);
    }
    return GetDIBImage(GetCurrentImageIndex());
}

/*****
*
*   Save DICOM document in different formats
*
*****
void CDCMDoc::OnFileSaveAs()
{
    BYTE    format;
    CString ffilter=_T( "DICOM (*) |*|"
        "Windows directory (*.bmp, *.txt, *.wav) |*.||");
    CString fname=DICOMDocument::GetFilename();

    // Display modal file dialog
    CFileDialog fd( FALSE, NULL, fname,
        OFN_HIDEREADONLY|OFN_NONETWORKBUTTON|OFN_OVERWRITEPROMPT,
        ffilter);
    if (fd.DoModal() != IDOK ) return;

    // Get save
    fname=fd.GetPathName();
    switch(fd.m_ofn.nFilterIndex)
    {
        case 1:     format=DICOMDocument::FormatDICOMModified;    break;
        case 2:     format=DICOMDocument::FormatWINDOWSMultimedia; break;
        default:    format=DICOMDocument::FormatDICOMOriginal;    break;
    }
    DICOMDocument::SaveDocument (fname,format);
    UpdateAllViews(NULL);
}

void CDCMDoc::OnFileSave()
{
    DICOMDocument::SaveDICOM();
    UpdateAllViews(NULL);
}

/*****
*
*   Reset Doc image zoom; return true if reset was possible
*
*****
bool CDCMDoc::OnZoom(UINT nID, Image *pBmp)
{
    switch(nID)
    {
        case ID_ZOOM_1_1: m_imageZoom=1.0;    break;
        case ID_ZOOM_1_2: m_imageZoom=2.0;    break;
        case ID_ZOOM_1_3: m_imageZoom=3.0;    break;
        case ID_ZOOM_1_4: m_imageZoom=4.0;    break;
        case ID_ZOOM_2_1: m_imageZoom=0.5;    break;
        case ID_ZOOM_3_1: m_imageZoom=1.0/3;  break;
        case ID_ZOOM_4_1: m_imageZoom=0.25;   break;
        case 1: // next smaller zoom
            if(m_imageZoom<0.2) return false;
            m_imageZoom /= 1.1;
            break;
        case 2: // next larger zoom
            if(m_imageZoom>4) return false;
            m_imageZoom *= 1.1;
            break;
    }
}

```

```

default: return false; // skip out invalid zoom maps
}
pBmp->m_ScreenMap.ValidateZoom(m_imageZoom);
m_Lupa.Initialize(m_Lupa.l_scnSize, m_imageZoom, m_Lupa.l_zoom);
return true;
}
/*****
*
*   Set zoom to fit the entire application screen
*
*****/
void CDCMDoc::SetFitScreenZoom()
{
    if(GetNumberOfImages()<=0) return;
    int img_num=GetCurrentImageIndex(); // save current image index
    Image *img = DICOMDocument::GetImage(0);
    CRect main_client;
    AfxGetMainWnd()->GetClientRect(main_client);
    double screen_width=main_client.Width()-16;
    double screen_height=main_client.Height()-100;
    if(screen_width<16 && screen_height<16) return;
    m_imageZoomFitScreen=min(screen_width/img->GetWidth(),
                             screen_height/img->GetHeight());
    if(m_imageZoomFitScreen>4.0) m_imageZoomFitScreen=4.0;
    m_imageZoom=m_imageZoomFitScreen;

    img->m_ScreenMap.ValidateZoom(m_imageZoom);
    m_Lupa.Initialize(m_Lupa.l_scnSize, m_imageZoom, m_Lupa.l_zoom);

    int off_x=(int)(screen_width-m_imageZoomFitScreen*img->GetWidth())/2;
    int off_y=(int)(screen_height-m_imageZoomFitScreen*img->GetHeight())/2;

    // Reset all frames, browse included
    for(int n=0; n<GetNumberOfImages(); n++)
    {
        DICOMDocument::GetImage(n)->m_ScreenMap.SetLeftTopScreenPoint(off_x+4,off_y+4);
    }
    // Reset to original current image
    DICOMDocument::GetImage(img_num);
    UpdateAllViews(NULL);
}

void CDCMDoc::OnUpdateZoomFitScreen(CCmdUI* pCmdUI)
{
    CString zinfo;
    zinfo.Format("Fit view: %.0lf%%",100*m_imageZoomFitScreen);
    pCmdUI->SetText(zinfo);
    pCmdUI->Enable();
}

/*****
*
*   Launch Sound Recorder
*
*****/
void CDCMDoc::OnSoundRecord()
{
    SoundDialog sd;
    sd.DoModal();
    m_SoundFileName=sd.GetWavFileName();
}

/*****
*
*   Email document as attachment
*
*****/
void CDCMDoc::OnFileSendMail()
{
    CDocument::OnFileSendMail();
    /*
    Email mail;

```



```

CString message = CString("Attached is DICOM image \n");
mail.Send("", "DICOM image", message, //DICOMDocument::GetFileName()
"", AfxGetMainWnd()->GetSafeHwnd());
*/
}
void CDCMDoc::OnUpdateFileSendMail(CCmdUI* pCmdUI)
{
    CDocument::OnUpdateFileSendMail(pCmdUI);
}
/*****
*
*   Display DICOM header info
*
*****/
void CDCMDoc::OnViewDicomInfo()
{
    DICOMDocument::DisplayDICOMInfo();
}
/*****
*
*   Update toolbar image counter
*
*****/
void CDCMDoc::OnUpdateFrameNumber(CCmdUI* pCmdUI)
{
    m_MainFrame->ShowFrameNumber(GetCurrentImageIndex()+1);
}
/*****
*
*   Get pointer to the current view
*
*****/
CView* CDCMDoc::GetViewPtr()
{
    POSITION pos = GetFirstViewPosition();
    if(pos != NULL) return GetNextView(pos);
    else return NULL;
}
/*****
*
*   Redraw current view
*
*****/
void CDCMDoc::OnViewRefresh(BOOL erase_background/*=FALSE*/)
{
    CDCMView* pView = (CDCMView*)GetViewPtr();
    if(!pView) return;
    pView->OnViewRefresh(erase_background);
}

```

```

// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////

#ifndef AFX_MAINFRM_H__INCLUDED_
#define AFX_MAINFRM_H__INCLUDED_

#ifdef _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CMDIFrameWnd
{
    DECLARE_DYNAMIC(CMainFrame)
public:
    CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CMainFrame)
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL DestroyWindow();
//}}AFX_VIRTUAL

// Implementation
public:
    void SetDictionaryStatus(bool enabled);
    void EnableLogoAnimation(bool enable);
    void ShowMultiframeToolBar(bool show=true);
    void ShowFrameNumber(int n);
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected: // control bar embedded members
    CStatusBar m_StatusBar;
    IEToolBar m_ToolBarBasic, m_ToolBarMultiframe;
    CAnimateCtrl m_Animate;
    CReBar m_ReBar;
    CDialogBar m_wndDlgBar;

// Generated message map functions
protected:
    {{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnViewToolBar();
    afx_msg void OnUpdateViewToolBar(CCmdUI* pCmdUI);
    afx_msg void OnDropFiles(HDROP hDropInfo);
    }}}AFX_MSG
    afx_msg void OnUpdateProgressStatus(CCmdUI *pCmdUI);
    afx_msg void OnToolBarDropDown(NMTOOLBAR* pnmh, LRESULT* plRes);
    DECLARE_MESSAGE_MAP()
private:
    bool m_showToolbars;
    CString m_paneString;
};

/////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous
// line.

#endif // !defined(AFX_MAINFRM_H__INCLUDED_)

```